BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

Docket No.: 56937-100

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of	:	Customer Number: 20277
Kei YONEDA, et al.	:	Confirmation Number: 4830
Serial No.: 10/727,055	:	Group Art Unit: 2122
Filed: December 04, 2003	:	Examiner:

For: SOFTWARE PROCESSING METHOD AND SOFTWARE PROCESSING SYSTEM

TRANSMITTAL OF CERTIFIED PRIORITY DOCUMENT(S)

Mail Stop CPD
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

At the time the above application was filed, priority was claimed based on the following application:

Japanese Patent Application No. JP 2002-355683, filed on December 6, 2002.

A copy of each priority application listed above is enclosed.

Respectfully submitted,

MCDERMOTT, WILL & EMERY

Michael E. Fogarty
Registration No. 36,139

600 13th Street, N.W.
Washington, DC 20005-3096
(202) 756-8000 MEF:gav
Facsimile: (202) 756-8087
Date: May 4, 2004

10/727,055
December 4, 2003
Kei YOKEDA, et al.

日本国特許庁
JAPAN PATENT OFFICE

McDermott, Will & Emery

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００２年１２月　６日

出願番号
Application Number: 特願２００２－３５５６８３

[ST. 10/C]: [ＪＰ２００２－３５５６８３]

出願人
Applicant(s): 松下電器産業株式会社

２００３年１１月１７日

特許庁長官
Commissioner,
Japan Patent Office

今井康夫

出証番号　出証特２００３－３０９４７７８

【書類名】　特許願

【整理番号】　5037740044

【あて先】　特許庁長官　殿

【国際特許分類】　G06F 13/00

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　米田　圭

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　河本　功

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　喜田　誠司

【発明者】

【住所又は居所】　大阪府門真市大字門真１００６番地　松下電器産業株式会社内

【氏名】　松林　貴明

【特許出願人】

【識別番号】　000005821

【氏名又は名称】　松下電器産業株式会社

【代理人】

【識別番号】　100086737

【弁理士】

【氏名又は名称】　岡田　和秀

【電話番号】　06-6376-0857

【手数料の表示】

【予納台帳番号】　007401

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【包括委任状番号】　9305280

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　ソフトウェア処理方法およびソフトウェア処理システム

【特許請求の範囲】

【請求項１】　プロセッサが使用するリソースの使用状況を監視する使用状況監視処理の過程と、

前記使用状況監視処理の過程で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程とを含むことを特徴とするソフトウェア処理方法。

【請求項２】　前記ソフトウェア処理変更処理の過程は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記使用状況監視処理の過程で得られる競合情報に応じて、前記複数の実行方法の中から１つを選択することを特徴とする請求項１に記載のソフトウェア処理方法。

【請求項３】　前記リソースは、処理の記憶装置および前記プロセッサと前記記憶装置を接続するバスであり、

前記使用状況監視処理の過程は、前記バスの使用状況を監視することを特徴とする請求項１または請求項２に記載のソフトウェア処理方法。

【請求項４】　前記使用状況監視処理の過程は、複数クロック遡ったバスの使用状況を記憶し、過去および現在の使用状況から前記競合情報を生成することを特徴とする請求項３に記載のソフトウェア処理方法。

【請求項５】　前記使用状況監視処理の過程は、前記バスの使用時における使用時間を記憶し、前記使用時間が所定値以上か否かに基づき前記競合情報を生成することを特徴とする請求項３に記載のソフトウェア処理方法。

【請求項６】　前記リソースは、前記プロセッサからの処理要求に応じて処理を行う第２のプロセッサであって、

前記使用状況監視処理の過程は、前記第２のプロセッサの使用状況の監視を行うことを特徴とする請求項１に記載のソフトウェア処理方法。

【請求項７】　請求項６に記載のソフトウェア処理方法が、さらに、同一アドレスでアクセス可能な複数のメモリバンクを有し、前記使用状況監視処理の過程で得られる競合情報は、前記複数のメモリバンクのうち１つのメモリバンクの選

択を示す信号であることを特徴とする請求項６に記載のソフトウェア処理方法。

【請求項８】　請求項１に記載のソフトウェア処理方法がさらにソフトウェアのコンパイラを有し、

前記コンパイラは、

ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、

前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、

前記使用状況監視処理の過程で得られる競合情報により使用状況を判定する使用状況判定処理と、

前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することを特徴とする請求項１に記載のソフトウェア処理方法。

【請求項９】　請求項１に記載のソフトウェア処理方法がさらにソフトウェアのコンパイラを有し、

前記コンパイラは、

ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、

前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、

現時刻の前記使用状況監視処理の過程で得られる競合情報を記憶する記憶処理と、

過去の時刻における前記競合情報により使用状況を判定する使用状況判定処理と、

前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することを特徴とする請求項１に記載のソフトウェア処理方法。

【請求項１０】　前記競合情報は、前記リソースの処理要求が発行されてから処理が終了するまでの処理時間であり、

前記使用状況判定処理は、前記処理時間をある設定値と比較する処理であることを特徴とする請求項８または請求項９に記載のソフトウェア処理方法。

【請求項１１】　前記競合情報は、前記リソースの処理要求が発行されてから処理が実行するまでの待ち時間であり、

前記使用状況判定処理は、前記待ち時間をある設定値と比較する処理であることを特徴とする請求項８または請求項９に記載のソフトウェア処理方法。

【請求項１２】　前記使用状況判定処理は、定期的または不定期に前記リソースの使用状況の判定を見直すことを特徴とする請求項８から請求項１１までのいずれかに記載のソフトウェア処理方法。

【請求項１３】　前記使用状況判定処理は、乱数を用いて前記リソースの使用状況の判定を見直すことを特徴とする請求項１２にソフトウェア処理方法。

【請求項１４】　請求項９に記載のソフトウェア処理方法において、

前記コンパイラは、前記処理識別手段により抽出される処理が前記ソフトウェアの複数の箇所から抽出される場合、さらに、前記処理識別手段で識別された処理の出現箇所を識別する出現箇所識別処理を前記ソフトウェアに追加し、前記記憶処理は前記出現箇所毎に前記競合情報を記憶し、前記使用状況判定処理は前記出現箇所毎に記憶した前記競合情報を用いて判定を行うことを特徴とする請求項９に記載のソフトウェア処理方法。

【請求項１５】　プロセッサと、

前記プロセッサが使用するリソースと、

前記リソースの使用状況を監視する使用状況監視装置と、

前記使用状況監視装置で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更装置を有することを特徴とするソフトウェア処理システム。

【請求項１６】　前記ソフトウェア処理変更装置は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記競合情報に応じて、前記複数の実行方法の中から１つを選択することを特徴とする請求項１５に記載のソフトウェア処理システム。

【請求項１７】　前記リソースは、前記プロセッサからの処理要求に応じて処理を行う第２のプロセッサであって、

前記使用状況監視装置は、前記第２のプロセッサの使用状況の監視を行うこと

を特徴とする請求項１５に記載のソフトウェア処理システム。

【請求項１８】　前記競合情報は、前記プロセッサへの割り込み信号であることを特徴とする請求項１７に記載のソフトウェア処理システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、プロセッサと共有リソースを有するシステムにかかわり、特には、共有リソース競合時の処理速度の低下を防止する技術に関するものである。

【０００２】

【従来の技術】

共有リソースを有するシステムとしては様々なものがある。マルチプロセッサシステム、例えばＣＰＵのようなプロセッサとＤＳＰ（デジタルシグナルプロセッサ）で構成されるマルチプロセッサシステムにおいては、処理をプロセッサとＤＳＰで分担して実行することにより処理性能の向上を図りながら、メモリ、バス、周辺回路などを共有リソースとして共用することにより、チップ面積を縮小し、低コストを実現している。共有リソースとしては、通常、メモリ、バス、周辺回路などがあり、システムの目的、仕様に応じて様々である。

【０００３】

共有リソースに対してアクセス競合が発生した場合、例えば、プロセッサが外部メモリへのデータ読み出し中に、ＤＳＰからも同一のメモリに対してデータ読み出しの処理が発行された場合、バス調停回路によりアクセス競合に対する処理が制御される。バス調停回路の制御方法としては、ＤＳＰからのデータ読み出し処理は受け付けるが、プロセッサからのデータ読み出しが終了するまではＤＳＰからのデータ読み出しの処理は待たせる、あるいは実行中のプロセッサからのデータ読み出しをキャンセルし、ＤＳＰからのデータ読み出しを優先に実行し、ＤＳＰでの処理の終了後、プロセッサからのデータ読み出し処理を再実行する、といった具合に、処理の仕様に応じて様々である。

【０００４】

さらに、プロセッサとＤＳＰで構成されるマルチプロセッサシステムにおいて

、ＤＳＰはプロセッサからの処理要求を受け付けることにより処理を実行する場合、プロセッサとＤＳＰとの間の制御方式が突き放し制御方式であれば、ＤＳＰを共有リソースということができる。突き放し制御方式におけるＤＳＰは、プロセッサからの処理要求をＤＳＰの状態に関係なく受け付けるが、処理の実行に関してはＤＳＰの状態を判断した後で開始する。つまり、ＤＳＰが別の処理を実行中であれば、処理要求を受け付けてもすぐには処理を開始させずにしばらく待たせ、ＤＳＰ側でプロセッサからの処理要求が実行できる状態になった時点で自律的に処理を実行し、処理結果をプロセッサ側へ返す。この制御方式では、ＤＳＰまたは別回路などの調停回路においてプロセッサが発行する処理をスケジューリングすることにより、複数の処理が共通のＤＳＰを競合することなく使用することになる。したがって、この場合、ＤＳＰは共有リソースであるということができる。

【０００５】

【非特許文献１】

斎藤忠夫・発田弘著、「高性能コンピュータアーキテクチャ」、丸善株式会社、ｐ２９０、ｐ３０９

【０００６】

【発明が解決しようとする課題】

しかしながら、従来の共有リソースを有するシステムでは、共有リソースへのアクセス競合が発生した場合、調停回路により競合リソースを使用する処理を適当にスケジューリングするが、優先順位の低い処理は先の処理が終了するまで待つことになり、共有リソースへのアクセス競合が頻繁に発生するような場合、システム全体の処理速度が低下する要因となる。

【０００７】

さらに、調停回路は共有リソースの状態のみを把握しており、システム全体の状態を把握していない。例えばＤＳＰが他の処理を実行中にプロセッサからＤＳＰへの処理要求が発行された場合、従来の調停回路では、先の処理が終了するまでその処理を待たせる。一方、プロセッサはＤＳＰへ発行した処理の結果を受けて次の処理を実行するため、プロセッサも待ち状態となる場合がある。このよう

な状況では、先の処理が終了するまでＤＳＰへ発行された処理およびその発行元であるプロセッサが待ち状態となり、プロセッサおよびＤＳＰを効率良く使用することができない。

【０００８】

また、上記従来の共有リソースを有するシステムでは、ＤＳＰへの処理要求が重複するＤＳＰ競合を避けるためにソフトウェアをＣＰＵとＤＳＰに適切に割り振る必要がある。この割り振り作業は人手で行っており、ソフトウェアの修正が発生する毎に多大な時間を必要とし、工数がかかるという問題がある。

【０００９】

【課題を解決するための手段】

上記の課題を解決するために、本発明は次のような手段を講じる。

【００１０】

本発明によるソフトウェア処理方法は、プロセッサとプロセッサが使用するリソースで構成されるシステムに対し、プロセッサが使用するリソースの使用状況を監視する使用状況監視処理の過程と、前記使用状況監視処理の過程で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程とを含むものである。

【００１１】

上記のソフトウェア処理方法に対応する本発明によるソフトウェア処理システムは、プロセッサと、前記プロセッサが使用するリソースと、前記リソースの使用状況を監視する使用状況監視装置と、前記使用状況監視装置で得られる競合情報に応じて、実行されるソフトウェアの処理方法を変更するソフトウェア処理変更装置を有するものである。

【００１２】

上記構成によれば、リソースの競合を動的に判定し、リソースを使用するソフトウェアの処理方法を変更することで、リソース競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【００１３】

上記において、前記ソフトウェア処理変更処理の過程、または、前記ソフトウ

ェア処理変更装置についての１つの態様は、ソフトウェアがある処理を行うのに複数の実行方法を持ち、前記ソフトウェアの実行中に前記使用状況監視処理の過程で得られる競合情報に応じて、前記複数の実行方法の中から１つを選択するものである。

【００１４】

上記において、前記リソースが、処理の記憶装置および前記プロセッサと前記記憶装置を接続するバスである場合において、前記使用状況監視処理の過程についての１つの態様は、前記バスの使用状況を監視するものである。これによれば、バスアクセス競合の発生を軽減し、バスを効率良く使用することが可能となる。したがって、バスアクセスの競合の発生によるシステムの処理速度の低下の影響を軽減することができる。

【００１５】

上記において、前記使用状況監視処理の過程についての１つの態様は、複数クロック遡ったバスの使用状況を記憶し、過去および現在の使用状況から前記競合情報を生成することである。また、上記において、前記使用状況監視処理の過程についての別の態様は、前記バスの使用時における使用時間を記憶し、前記使用時間が所定値以上か否かに基づき前記競合情報を生成することである。クロック数または時間を指標にして、バスの使用状況を容易に判定することができる。

【００１６】

上記において、前記リソースが、前記プロセッサからの処理要求に応じて処理を行う第２のプロセッサである場合において、前記使用状況監視処理の過程、または、前記使用状況監視装置についての１つの態様は、前記第２のプロセッサの使用状況の監視を行うことである。この場合、第２のプロセッサを使用しているときに別の処理が発生した場合、前記競合情報として前記プロセッサへの割り込み信号を用いて、割り込みを発生させる割り込み機能を備えるとよい。

【００１７】

上記のソフトウェア処理方法が、さらに、同一アドレスでアクセス可能な複数のメモリバンクを有する場合において、前記使用状況監視処理の過程で得られる競合情報についての１つの態様は、前記複数のメモリバンクのうち１つのメモリ

バンクの選択を示す信号である。この場合、第２のプロセッサでの処理の実行毎に、あるいは第２のプロセッサで実行された処理が終了する毎に、複数のメモリバンクの中から１つを選択する選択信号を切り替える機能を備えるとよい。

【００１８】

上記によれば、プロセッサの競合を動的に判定し、割り込みを行ったり、メモリバンクを選択することで、プロセッサ競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【００１９】

上記のソフトウェア処理方法が、さらにソフトウェアのコンパイラを有する場合において、前記コンパイラについての１つの態様は、ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、前記使用状況監視処理の過程で得られる競合情報により使用状況を判定する使用状況判定処理と、前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することである。これにより、リソース競合を避け、システムの処理速度の低下を避ける。

【００２０】

また、上記のソフトウェア処理方法が、さらにソフトウェアのコンパイラを有する場合において、前記コンパイラについての１つの態様は、ソフトウェアから前記リソースを使用する処理を識別する処理識別手段と、前記処理識別手段により識別された処理と等価な、前記リソースを使用しない等価処理と、現時刻の前記使用状況監視処理の過程で得られる競合情報を記憶する記憶処理と、過去の時刻における前記競合情報により使用状況を判定する使用状況判定処理と、前記使用状況判定処理の結果に応じて、前記処理識別手段により識別された処理を前記等価処理に置き換える置換処理とを、前記ソフトウェアに追加することである。これによれば、競合の履歴を加味した処理となり、リソース競合の判定を高精度に行うことができる。

【００２１】

上記において、前記競合情報についての１つの態様は、前記リソースの処理要

求が発行されてから処理が終了するまでの処理時間であり、この場合に、前記使用状況判定処理は、前記処理時間をある設定値と比較する処理を行うものである。また、上記において、前記競合情報についての別の態様は、前記リソースの処理要求が発行されてから処理が実行するまでの待ち時間であり、この場合に、前記使用状況判定処理は、前記待ち時間をある設定値と比較する処理を行うものである。これによれば、処理の待ち時間や処理時間などを計測することでリソース競合が発生中か否かを判定し、リソース競合が発生中ならば新たにリソース競合を引き起こす処理を発行しない。これにより、競合によるシステムの処理速度の低下を抑制することが可能である。

【００２２】

上記において、前記使用状況判定処理についての１つの態様は、定期的または不定期に前記リソースの使用状況の判定を見直すことである。この場合に、乱数を用いて前記リソースの使用状況の判定を見直すのがよい。これによれば、定期的または不定期に競合判定を補正するので、さらに高精度な競合判定が可能である。

【００２３】

上記のソフトウェア処理方法において、前記コンパイラにおける前記処理識別手段により抽出される処理が前記ソフトウェアの複数の箇所から抽出される場合、前記コンパイラについての１つの態様は、さらに、前記処理識別手段で識別された処理の出現箇所を識別する出現箇所識別処理を前記ソフトウェアに追加し、前記記憶処理は前記出現箇所毎に前記競合情報を記憶し、前記使用状況判定処理は前記出現箇所毎に記憶した前記競合情報を用いて判定を行うことである。これによれば、競合情報を競合を引き起こす処理の出現箇所とともに記憶することで、ループ処理における競合判定を高精度に行うことが可能である。

【００２４】

以上のようにして、本発明によれば、リソースの競合およびプロセッサ上で実行する処理の競合を考慮した人手での処理の割り当てを行うことがなくなり、工数の削減が図れる。

【００２５】

【発明の実施の形態】

図１は、本発明の実施の形態にかかわるソフトウェア処理システムの基本構成を示すブロック図である。

【００２６】

図１において、プロセッサ１０１がソフトウェア処理の実行を始めると、使用状況監視処理手段１０３は、プロセッサ１０１が使用するリソース１０２の使用状況を監視し、その出力を受けて、リソース１０２の使用状況の情報を取得する。リソース１０２の使用状況の情報の結果に応じて、ソフトウェア処理変更処理手段１０４は実行されるソフトウェアの処理方法を変更する。

【００２７】

なお、前記使用状況監視処理手段１０３および前記ソフトウェア処理変更処理手段１０４は、それぞれソフトウェアで実現したものを想定しているが、それぞれハードウェアで実現しても良く、少なくとも、リソースの使用状況を監視し、その情報を取得する機能、および、前記情報の出力を受けてソフトウェアの処理方法を変更する機能を有していれば良いものである。

【００２８】

以下、本発明の第１から第６までの実施の形態について、図を用いて説明する。

【００２９】

（第１の実施の形態）

以下、本発明の第１の実施の形態を図面に基づいて説明する。

【００３０】

図２は、本発明の第１の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。

【００３１】

図２において、１００１はＣＰＵ、１００２はＣＰＵ１００１が使用するバス、１００３はＣＰＵ１００１の周辺回路、１００４は外部メモリ、１００５はバス１００２上のバスアクセスの状態を記憶する使用状況監視装置である。

【００３２】

図３は、本発明の第１の実施の形態にかかわるソフトウェア処理システムのバス１００２の使用状況監視装置１００５の構成を示す図である。

【００３３】

図３において、１１０１はバス１００２上でバスアクセスが起きているかどうかを特定の期間だけ識別するバスアクセス識別装置、１１０２はバスアクセス識別装置１１０１の出力を受けて、バスアクセスの状態を記録するバスアクセス状態レジスタである。

【００３４】

図４は、本発明の第１の実施の形態にかかわるソフトウェア処理システムのソフトウェアの構成を示す図である。

【００３５】

図４において、１２０１はＣＰＵ１００１上で動作するプログラム、１２０２，１２０３，１２０４はプログラム１２０１中に記述される処理、１２０５，１２０６，１２０７は処理１２０２，１２０３，１２０４とそれぞれ同等の機能を持ち、バス１００２を使用しないＣＰＵ１００１用の処理である。

【００３６】

図５は、本発明の第１の実施の形態にかかわるソフトウェア処理システムにおけるコンパイルフローを示すフローチャートである。

【００３７】

図５において、１３０１はコンパイラがプログラム１２０１をコンパイルするコンパイル手段、１３０２はプログラム１２０１に記述される処理において、バス１００２を使用する処理を識別する処理識別手段、１３０３は処理識別手段１３０２の出力を受けて、使用状況監視装置１００５が持つバスアクセス状態レジスタ１１０２の値を読み取り、バスアクセスの情報を動的に判定するプログラムである使用状況判定処理、および、処理識別手段１３０２により識別された処理を、処理識別手段１３０２により識別された処理と等価な前記リソースを使用しない等価処理に置き換える置換処理を付加する処理付加手段である。

【００３８】

図６は、本発明の第１の実施の形態にかかわるソフトウェア処理システムにお

けるバスアクセス識別装置１１０１が識別するバスアクセスの状態を示す図であり、図７は、そのソフトウェア処理システムにおける処理付加手段１３０３によるソフトウェアの変更を示す図である。

【００３９】

以下、第１の実施の形態のソフトウェア処理システムの動作について説明する。

【００４０】

本実施の形態においては、ＣＰＵ１００１によって、プログラム１２０１に記述される処理１２０２，１２０３，１２０４が、処理１２０２，処理１２０３，処理１２０４の順番に処理され、かつ、処理１２０２，１２０３はバス１００２を使用する。

【００４１】

図４、図５に示すように、プログラム１２０１のコンパイルの際には、まず、処理１２０２，１２０３，１２０４はコンパイル手段１３０１によってコンパイルされる。また同時に、処理識別手段１３０２によってバス１００２を使用する処理として処理１２０２，１２０３が識別される。次に、処理識別手段１３０２によって得られた処理１２０２と処理１２０３の実行の直前に使用状況判定処理および置換処理が行われるように、処理付加手段１３０３によって使用状況判定処理および置換処理が付加される。前記付加処理により、プログラム１２０１の処理は、処理１２０２が行われる直前において、使用状況判定処理および置換処理が実行されることとなる。

【００４２】

バスアクセス状態レジスタ１１０２に記録される値は、バスアクセス識別装置１１０１によって算出される。この算出される値は、図６に示すように、バスアクセスが発生していれば増加し、バスアクセスが発生していなければ減少し、値の増減の度合いは任意に設定することができる。

【００４３】

ここで、算出される値をαとし、αの増減の度合いをβとする。αの算出方法は、まず、αに１サイクル毎にβを掛けて、次に、その値（$\alpha \times \beta$）に、バスア

クセスが発生していれば１を、バスアクセスが発生していなければ０を足す。つまり、ｎサイクル目でのαの値をα（ｎ）とすると、

α（ｎ）＝β×α（ｎ－１）＋（１または０）

となる。例えば、βを０．９とした場合、αの値は０から１０の間の値をとることになり、そのαの値がバスアクセスが頻繁に発生しているかどうかの目安の値となる。

【００４４】

βを０．９とした場合のバスアクセスの状態を表すαとサイクル数の関係は図６のように示される。αの値の変化が増加している箇所はバスサイクルが頻繁に発生しており、αの値が減少している箇所はバスアクセスが発生していない。

【００４５】

以上のように、バスアクセス識別装置１１０１は、バスアクセスが頻繁に発生しているかどうかの目安となるαの値を算出し、その結果をバスアクセス状態レジスタ１１０２へ記録する。

【００４６】

使用状況判定処理は、バスアクセス状態レジスタ１１０２に記録されている値αを読み取り、特定の値と比較してバスアクセスの状態を判定する。この特定の値は任意に設定することができる値であり、バスアクセスが頻繁に発生しているかどうかを判定する閾値となる。前述のβ＝０．９の場合、αは０から１０の値をとることになるので、前述の閾値を５と設定したとすると、αが５より大きければ、その時点でバスアクセスが頻繁に発生していると判定できる。逆にαが５以下であれば、バスアクセスは頻繁に発生していないと判定できる。

【００４７】

この判定の結果を受けて、置換処理は、処理１２０２を処理１２０５へ変更する。つまり、判定の結果、バス１００２が頻繁に使用されていれば、置換処理によって処理１２０２の代わりに処理１２０５が行われる。逆に、判定の結果、バス１００２が頻繁に使用されていなければ、処理１２０２がそのまま行われる。

【００４８】

処理識別手段１３０２によってバス１００２を使用する処理であると識別され

た処理１２０３についても同様に、処理付加手段１３０３によって付加された使用状況判定処理および置換処理が行われ、バス１００２が頻繁に使用されていれば、処理１２０３の代わりに処理１２０６が行われ、バス１００２が頻繁に使用されていなければ、処理１２０３が行われる。

【００４９】

以上のように本実施の形態によれば、使用状況判定処理および置換処理によって、バス１００２におけるバスアクセス競合の発生を軽減し、バス１００２を効率良く使用することが可能となり、バスアクセスの競合の発生によるシステムの処理速度の低下の影響を軽減することができる。

【００５０】

なお、本実施の形態では、プロセッサを１つのＣＰＵ、リソースを１つのバスとしたが、それぞれ複数のＣＰＵ、バスでも良く、少なくとも１つのリソースとそのリソースを使用する少なくとも１つのプロセッサであれば良いものである。

【００５１】

なお、本実施の形態では、バスアクセス状態レジスタにαを求めているが、αの求め方あるいはバスのアクセス頻度を求める方法は本方法によらずともよく、指標として求めることができるものであれば同等の効果を得られる。

【００５２】

（第２の実施の形態）

第２の実施の形態によるソフトウェア処理システムを図８～図１５を用いて具体的に説明する。

【００５３】

図８は、第２の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図８においてソフトウェア処理システムは、プロセッサ２００１と、ＤＳＰ２００２と、プロセッサ２００１とＤＳＰ２００２で実行するソフトウェアを格納した記憶装置２００３と、周辺回路２００４と、バス調停回路２００５と、外部メモリ２００６と、共有バス２００７で構成される。

【００５４】

図８では、プロセッサ２００１はＤＳＰ２００２を突き放し制御方式で制御しており、プロセッサ２００１が記憶装置２００３から読み出したソフトウェアがＤＳＰ２００２で実行すべき処理の場合は、プロセッサ２００１はＤＳＰ２００２へ処理要求を発行し、ＤＳＰ２００２は処理要求を受けた後、処理を実行する。外部メモリ２００６はプロセッサ２００１、ＤＳＰ２００２が処理を実行する際に使用する共有メモリである。バス調停回路２００５はバスアクセスを管理しており、共有メモリへのアクセス競合、すなわち共有バス２００７のバス競合が発生した場合には、外部メモリ２００６へアクセスする処理を実行するか否かを判定する。

【００５５】

ここで、バス調停回路２００５の制御フローを図９に示す。図９において、２１０１は共有メモリである外部メモリ２００６へのアクセス要求（読み出しまたは書き込み要求）が発生したか否かを判定するアクセス要求判定手段であり、２１０２は現在、外部メモリ２００６がアクセス中か否かを判定するメモリ状態判定手段であり、２１０３はアクセス要求に対しバスを解放しメモリアクセスを促すメモリアクセス許可手段であり、２１０４は処理が終了したか否かを判定する処理終了判定手段である。

【００５６】

外部メモリ２００６へのアクセス中に新たな外部メモリ２００６へのアクセス要求が発生した場合、つまりバス競合が発生した場合、バス調停回路２００５は、一旦、アクセス要求を受理し、実際の処理に関しては先の処理が終了するまで待機（ｗａｉｔ）状態となる。外部メモリ２００６へのアクセスが頻繁に発生すると、バス競合が発生し、バス調停回路２００５により処理が待機（ｗａｉｔ）状態となる。その結果、ソフトウェア処理システム全体の処理速度が低下する。そこで第２の実施の形態では、バス競合を避け、処理速度の低下を防ぐ。

【００５７】

本実施の形態におけるコンパイラは、処理記述言語で記述されたソフトウェアからオブジェクトコードを出力する通常のコンパイラとしての機能の他に、図１０の処理フローに示す機能を有する。コンパイラは、まず、処理識別手段２２０

１として外部メモリ２００６へ頻繁にアクセスする処理を識別する。これは、処理識別手段２２０１が外部メモリ２００６へアクセスする処理をあらかじめ認識していれば可能である。次に、識別した処理に対して、これと等価であるが、外部メモリ２００６へのアクセスが少ない処理である等価処理があらかじめライブラリ等で用意されているか否かを判定する。もし用意されていれば、コンパイラは処理識別手段２２０１で識別した処理に対して等価処理、記憶処理、使用状況判定処理、置換処理を追加する。ここで、記憶処理とは外部メモリ２００６へ頻繁にアクセスする処理の要求が発行されてから処理が終了するまでの処理時間を記録する処理であり、使用状況判定処理とは記憶処理で記憶した使用状況の情報からバス競合が発生しているか否かを判定する処理であり、置換処理とは処理識別手段２２０１で識別した処理を等価処理へ置き換える処理を示す。

【００５８】

図１１は、図１０の処理フローをもつコンパイラを使用したときの外部メモリ２００６へ頻繁にアクセスする処理の変更を示す図である。図１１では、外部メモリ２００６へ頻繁にアクセスする処理を関数“ｍｅｍ_ａｃｓ_ｍａｎｙ（）”とし、この等価処理を“ｍｅｍ_ａｃｓ_ｆｅｗ（）”としている。コンパイラは、あらかじめ外部メモリ２００６へアクセスする処理を登録することで、処理識別手段２２０１を用いてソフトウェアの中から関数“ｍｅｍ_ａｃｓ_ｍａｎｙ（）”を識別する。

【００５９】

次に、コンパイラは処理識別手段２２０１で識別した処理に対して上記の等価処理、記憶処理、使用状況判定処理、置換処理を付加する。

【００６０】

まず、図１１（ｂ）の２行目では、ソフトウェアに使用状況判定処理を付加する。ここで“ｔａｓｋ_ｔｉｍｅ”はプロセッサ２００１が外部メモリ２００６へ頻繁にアクセスする処理の要求を発行してから処理が終了するまでの処理時間であり、“ＤＥＦＩＮＥＤ_ＴＩＭＥ”はあらかじめ設定された設定値である。処理時間“ｔａｓｋ_ｔｉｍｅ”は、過去に発生した外部メモリ２００６へ頻繁にアクセスする処理の処理時間を表しており、現時刻に発生する外部メモリ２０

０６へ頻繁にアクセスする処理を実行する前に“ｔａｓｋ_ｔｉｍｅ”を参照し、設定値である“ＤＥＦＩＮＥＤ_ＴＩＭＥ”と比較することでバス競合を判定する。バス競合が発生中と判定された場合、図９の制御フローに従うバス調停回路２００５では、バス競合の発生時に処理を待機状態とするため、そのときの処理時間は長くなる。つまり、外部メモリ２００６へ頻繁にアクセスする処理の処理時間がある程度大きくなるとバス競合が発生したと推定できる。なお、設定値については、ソフトウェア処理システムの仕様に応じて的確な値を設定するのが望ましい。

【００６１】

次に、図１１（ｂ）の３行目から６行目は、記憶処理を示す。使用状況判定処理によりバス競合が発生していないと推定された場合、“ｍｅｍ_ａｃｓ_ｍａｎｙ（）”実行の前後のタイマーの値を参照し、その差を算出することで処理時間“ｔａｓｋ_ｔｉｍｅ”を算出している。

【００６２】

図１１（ｂ）の８行目から１０行目は、置換処理を示す。使用状況判定処理によりバス競合が発生していると推定された場合、さらにバス競合を発生させることを防ぐため、外部メモリ２００６へのアクセスが少ない処理に置き換える。なお、置換処理で置換する処理は、処理の整合性を保つために機能的に等価であるが、外部メモリ２００６へのアクセスが少ない処理が望ましい。

【００６３】

第２の実施の形態では、上述のコンパイラが追加した等価処理、記憶処理、使用状況判定処理、置換処理により、使用状況監視処理の過程およびソフトウェア処理変更処理の過程を実現する。つまり、使用状況監視処理の過程とは、記憶処理により処理時間“ｔａｓｋ_ｔｉｍｅ”を記憶しており、使用状況判定処理へ処理時間“ｔａｓｋ_ｔｉｍｅ”を出力する。ソフトウェア処理変更処理の過程とは、識別した処理を等価処理へ置き換える置換処理であり、使用状況監視処理の過程から出力される処理時間に基づいてバス競合を判定し、バス競合が発生していれば処理の置き換えを行う。

【００６４】

ここで、本実施の形態によるバス競合の判定フローを図１２に示す。処理識別手段２２０１で識別した処理に対し、前回識別した処理を実行した際の処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”を比較する。処理時間“ｔａｓｋ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合には、使用状況判定処理はバス競合が発生中であると判定し、等価処理への置き換えを行う。逆に処理時間“ｔａｓｋ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”よりも小さい場合には、使用状況判定処理はバス競合が発生していないと判定し、処理識別手段２２０１で識別した処理を、処理時間“ｔａｓｋ_ｔｉｍｅ”の記憶を行いながら実行する。

【００６５】

ここで、「前回識別した処理を実行した際の処理時間」とは、２つ考えられる。１番目の処理時間は、処理識別手段２２０１で識別した処理の中で、現時刻から１回前に記録した処理時間、つまり、記録した処理時間の中で最も新しいものが該当する。２番目の処理時間は、現時刻で識別した処理と同一の処理を過去に実行したときに記録した処理時間が該当する。１番目の処理時間では、処理時間は常に最新のものを１つだけ記憶するが、２番目の処理時間では、識別された処理毎に処理時間を記憶する必要がある。

【００６６】

以上、本実施の形態では、処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比較結果より、バス競合が発生中か否かを判定し、バス競合が発生中と判定された場合は、外部メモリ２００６へ頻繁にアクセスする処理を外部メモリ２００６へのアクセスが少ない処理に自動で置換する。その結果、動的にバス競合を判定しながら自動で処理の割り当てを行うことができるとともに、新たなバス競合を避けることができ、処理速度の低下を防止できる。なお、第２の実施の形態における上記の発明内容を後述の発明と区別するために第１の手法とする。

【００６７】

さらに、第２の手法としての使用状況判定処理においては、バス調停回路２００５から出力された競合情報である処理時間“ｔａｓｋ_ｔｉｍｅ”と事前に設

定しておいた設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”との比較結果に加えて、乱数を使用することでバス競合の判定を行う。

【００６８】

図１３は、処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比に対する等価処理への置き換えを行う確率を示している。処理時間“ｔａｓｋ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”よりも小さい場合、言い換えれば、図１３で処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が１より小さい場合、使用状況判定処理は第１の手法と同様にバス競合が発生していないと判定し、等価処理の置き換えを行わない。

【００６９】

しかし、処理時間“ｔａｓｋ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合、言い換えれば、図１３で処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が１以上の場合、図１３の確率に従って等価処理への置き換えを行ったり、あるいは等価処理への置き換えは行わずに、処理時間を再計測し記憶する。これは、処理時間“ｔａｓｋ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合、第１の手法では必ずバス競合が発生中と判定していたが、第２の手法では乱数を用いた確率に応じてバス競合を判定するので、第１の手法とは逆の判定結果を出力する可能性を持たせた。逆の判定結果とは、バス競合が発生していないという判定結果である。

【００７０】

そして、本実施の形態では図１２に示すバス競合判定フローに従い、処理時間を再度記録することになる。つまり、バス競合の判定に乱数を加えることで判定を見直しする。これにより、実際にはバス競合が解消されたにもかかわらず、使用状況判定処理はバス競合がそのまま継続されている、といった間違った判断を避けることができる。

【００７１】

第２の手法は一連の処理の中でバス競合が頻繁に発生するシステムに対して有効である。なお、図１３では、処理時間“ｔａｓｋ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が大きければ大きいほど、つまり、処理時間“ｔａ

ｓｋ_ｔｉｍｅ”が設定値よりも大きな値をとればとるほど等価処理へ置き換える確率が小さくなっている。これは、処理時間“ｔａｓｋ_ｔｉｍｅ”が長ければ長いほど近い未来にバス競合は解消されるということが一般的に考えられるからである。

【００７２】

なお、図１３の発生確率の値については、対象システムの仕様に応じて適当な値を与える必要がある。

【００７３】

以上、第２の手法により使用状況判定処理におけるバス競合の判定精度を向上できる。

【００７４】

なお、本実施の形態の第２の手法では、バス競合の判定について乱数を用いた確率を使用したが、乱数を用いないバス競合の判定も考えられる。例えば、バス競合時における処理時間の平均値または最大値などの定数があらかじめ既知であれば、バス競合中と判定されてから一定周期ごとに処理時間の再計測及び記録を行うことも可能である。

【００７５】

次に、第３の手法について説明する。

【００７６】

外部メモリ２００６へ頻繁にアクセスする処理が複数の箇所に存在する場合、その処理の呼び出し元（出現箇所）別にバス競合を判定すれば判定精度は向上する。第３の方法では、処理識別手段２２０１で識別した処理の出現箇所毎に処理時間を記憶し、それを使用することでバス競合の判定を高精度に行う。

【００７７】

図８のソフトウェア処理システムでは、複数の処理が記述された関数を実行する際には、現在のプログラムカウンタを退避レジスタに一時的に退避させた後、関数の処理を実行し、関数の処理が終了した後、退避レジスタからプログラムカウンタを読み出してプログラムカウンタを元の値に戻す操作を行う。第３の手法では、出現箇所識別処理が上述の退避レジスタからプログラムカウンタを読み出

すことで処理識別手段２２０１で識別した処理の出現箇所を特定する。

【００７８】

図１２の判定フローをもつ第１の手法に、さらに出現箇所識別処理を追加した場合のバス競合の判定フローを図１４に示す。

【００７９】

まず、処理識別手段２２０１で識別された処理に対し、出現箇所識別処理により処理の出現箇所を調査する。上述したように退避レジスタを参照することで、出現箇所の特定は可能である。次に、出現箇所毎に記憶した処理時間を参照する。次に、出現箇所毎に記憶した処理時間を設定値と比較することでバス競合が発生中か否かを判定し、バス競合が発生中であれば等価処理に置き換えて処理を実行し、バス競合が発生していなければ、処理時間を記憶しながら識別した処理をそのまま実行する。なお、出現箇所毎に処理時間を記憶する部分に処理時間が記憶されていない場合は、バス競合が発生していないと判定されたとみなし、処理時間を計測しながら処理識別手段２２０１で識別された処理を実行し、計測した処理時間を新規に記憶する。

【００８０】

第３の手法は、ループ処理など同一の処理が繰り返して処理される場合、出現箇所別に判定した処理時間を参照することでバス競合の判定精度が向上する。

【００８１】

なお、本実施の形態における処理時間“ｔａｓｋ_ｔｉｍｅ”を、第１の実施の形態の使用状況監視装置に記憶することでバス競合の判定を行うことも可能である。このときのバス競合の判定フローを図１５に示す。処理時間を求めた後、第１の実施の形態における使用状況監視装置にも処理時間を記憶することにより、バス競合を動的に判定でき、バス競合に伴う処理速度の低下を防ぐことが可能である。

【００８２】

（第３の実施の形態）

図１６は第３の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図１６において、ソフトウェア処理システム

は、ＤＳＰ４００１とプロセッサ４００２と、ＤＳＰ使用状況監視装置４００３と、ＤＳＰ４００１とプロセッサ４００２で実行する処理である図１７のオブジェクトコード４１０５を格納したメモリ４００４で構成される。メモリ４００４は、ＤＳＰ４００１とプロセッサ４００２が処理を実行する際の共有メモリとして処理結果や処理データを記憶する。

【００８３】

図１７は、本発明の第３の実施の形態にかかわるコンパイラの処理フローを示す。

【００８４】

図１８は、本発明の第３の実施の形態にかかわるコンパイラによる処理の変更を示す。

【００８５】

図１９は、本発明の第３の実施の形態にかかわるＤＳＰ競合の判定フローを示す。

【００８６】

図２０は、本発明の第３の実施の形態にかかわるソフトウェア処理システムにおける処理の流れを示す。

【００８７】

第３の実施の形態によるソフトウェア処理システムを図１６～図２０を用いて具体的に説明する。

【００８８】

図１６では、プロセッサ４００２はＤＳＰ４００１を突き放し制御方式で制御しており、プロセッサ４００２がメモリ４００４から読み出したソフトウェアがＤＳＰ４００１で実行する処理の場合は、プロセッサ４００２はＤＳＰ４００１へ処理要求を発行し、ＤＳＰ４００１は処理要求を受けた後、処理を実行する。ここで、ＤＳＰ使用状況監視装置４００３は、現在、ＤＳＰ４００１で処理が実行中か否かを記憶したレジスタ（記憶装置）であり、特定のアドレスが割り当てられているため、ソフトウェアの処理から読み出し可能である。

【００８９】

図１７はソフトウェアのコンパイラのフローである。Ｃ言語などの処理記述言語で書かれたソースコード４１０２は、プロセッサ用ライブラリ４１０３とＤＳＰ用ライブラリ４１０４を用いて、オブジェクトコード４１０５に変換される。その結果、オブジェクトコード４１０５には実行に必要なプロセッサ用ライブラリ４１０６とＤＳＰ用ライブラリ４１０７が含まれる。

【００９０】

本実施の形態のコンパイラは、処理記述言語からオブジェクトコードを生成する通常のコンパイラの機能の他に、ソースコード４１０２からＤＳＰ４００１で実行する処理を識別する処理識別手段と、処理識別手段で識別した処理に対し特定の処理を付加する処理付加手段を備える。処理識別手段および処理付加手段による処理の変更を図１８に示す（左の数字は行番号）。ここで、図１８（ａ）の処理記述言語で書かれたソースコードは、処理識別手段を用いて、ソースコード４１０２からＤＳＰ４００１で実行されるとして識別された処理を表している。つまり、関数“ｆｕｎｃ１（）”、“ｆｕｎｃ２（）”、“ｆｕｎｃ３（）”は全てＤＳＰ４００１で実行される処理である。なお、ＤＳＰ４００１で実行する処理をあらかじめ処理識別手段に登録することでソースコードからＤＳＰ４００１で実行する処理を識別可能である。

【００９１】

図１８（ｂ）の処理記述言語で書かれたソースコードは、処理付加手段で付加された処理を示す。なお、便宜上、図１８（ｂ）では、処理付加手段はｆｕｎｃ２のみに特定の処理を付加しているが、処理付加手段は、処理識別手段で識別した全ての処理に対して特定の処理を付加しており、ｆｕｎｃ１，ｆｕｎｃ３についてもｆｕｎｃ２と同様に特定の処理が付加される。

【００９２】

処理付加手段により付加される特定の処理とは、使用状況判定処理と置換処理である。使用状況判定処理とは、ＤＳＰ４００１が処理を実行中か否かをＤＳＰ使用状況監視装置４００３を読み出すことで判定する処理である。図１８（ｂ）では、８行目の“ｉｆ（ＤＳＰが待ち状態）”が使用状況判定処理に相当する。ここで待ち状態とは、ＤＳＰ４００１は処理を実行しておらず、プロセッサ４０

０２からの処理要求を待っている状態を表す。図１８（ｂ）では９行目から１１行目が相当し、ｆｕｎｃ２の処理内容として、ＤＳＰ用ライブラリ４１０４を使用した場合を“ｆｕｎｃ２_ｄｓｐ（）”、プロセッサ用ライブラリ４１０３を使用した場合を“ｆｕｎｃ２_ｃｐｕ（）”としている。図１８では、置換後の処理としてプロセッサで同等の機能をもつ処理を行った場合を示す。

【００９３】

つまり、本実施の形態では、ＤＳＰ４００１を使用する処理を識別し、その処理を実行する前にＤＳＰ使用状況監視装置４００３を観測することでＤＳＰ４００１の状態を判定する。そして、判定結果を受けて、ＤＳＰ４００１が待ち状態であれば、そのままＤＳＰ４００１で処理を実行するが、ＤＳＰ４００１が他の処理を実行中の場合は、ＤＳＰ４００１を使用しない他の処理に置換する。このようなソフトウェア処理変更処理により、ＤＳＰ競合を避けることができる。参考として、図１９に本実施の形態のＤＳＰ競合の判定フローを示す。

【００９４】

次に具体例として、本実施の形態のソフトウェア処理システムにおける処理の流れを図２０に示す。

【００９５】

図２０（ａ）は、ＤＳＰ競合が発生しない場合における本実施の形態のソフトウェア処理システムの処理の流れである。ソフトウェアには、ｆｕｎｃ０，ｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３，ｆｕｎｃ４，ｆｕｎｃ５，ｆｕｎｃ６の処理があり、ｆｕｎｃ０，ｆｕｎｃ２，ｆｕｎｃ４については、プロセッサ４００２からＤＳＰ４００１へ処理要求を発行し、それぞれＤＳＰ用ライブラリ４１０４を用いてｆｕｎｃ０_ｄｓｐ，ｆｕｎｃ２_ｄｓｐ，ｆｕｎｃ４_ｄｓｐをＤＳＰ４００１上で実行する。また、ＤＳＰ４００１とプロセッサ４００２は共有メモリを用いて互いの処理結果を受け渡しており、ｆｕｎｃ０_ｄｓｐとｆｕｎｃ２_ｄｓｐとｆｕｎｃ４_ｄｓｐとｆｕｎｃ５は処理結果を共有メモリ４００４に格納し、格納された処理結果は、ｆｕｎｃ３とｆｕｎｃ５が処理を実行する際に読み出す。図２０（ａ）のようにＤＳＰ競合が発生しない場合には、プロセッサ４００２とＤＳＰ４００１の資源を無駄なく使用しながら処理を実行する。

【００９６】

しかし、図２０（ｂ）のようにｆｕｎｃ０_ｄｓｐの処理時間が長くなった場合、ｆｕｎｃ１がｆｕｎｃ２_ｄｓｐの処理要求を発行した場合、ＤＳＰ競合が発生し、ｆｕｎｃ０_ｄｓｐが終了するまでｆｕｎｃ２_ｄｓｐは処理を開始できない。その結果、ｆｕｎｃ２_ｄｓｐの処理結果を用いるｆｕｎｃ３も実行できなくなり、処理全体の速度が遅くなってしまう。

【００９７】

そこで、本実施の形態のソフトウェア処理システムでは、ｆｕｎｃ１がｆｕｎｃ２_ｄｓｐの処理要求を発行した際に、ＤＳＰ使用状況監視装置４００３を観測することでＤＳＰ競合を認識し、ｆｕｎｃ２と等価な機能でありＤＳＰ４００１でなくプロセッサ４００２上で処理を行うｆｕｎｃ２_ｃｐｕをｆｕｎｃ２_ｄｓｐの代わりに実行する。つまり、図２０（ｂ）では、ＤＳＰ４００１で実行する処理の処理要求を発行する際にＤＳＰ４００１の状態を監視し、処理をＤＳＰ４００１かプロセッサ４００２に的確に割り振るソフトウェア処理変更処理を用いる。これによりＤＳＰ競合を防ぐことができ、処理速度の低下を防ぐ。さらに、処理を的確に割り当てて待ち状態のプロセッサ４００２を使用することで、資源を効率的に使用可能である。

【００９８】

なお、本実施の形態のコンパイラの処理識別手段、処理付加手段により、処理サイクル数は増えるが、ＤＳＰ使用状況監視装置４００３以外、新規にハードウェアを追加することなく、ＤＳＰ競合を避け、処理速度の低下を防止できる。

【００９９】

（第４の実施の形態）

第４の実施の形態によるソフトウェア処理システムを図２１～図２８を用いて具体的に説明する。

【０１００】

図２１は、第４の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図２１においてソフトウェア処理システムは、プロセッサ５００１と、ＤＳＰ５００２と、プロセッサ５００１とＤＳＰ５

００２で実行するソフトウェアを格納した記憶装置５００３と、周辺回路５００４と、バス調停回路５００７と、外部メモリ５００５と、共有バス５００６と、調停回路５００８と、使用状況監視装置５００９で構成される。図２１では、プロセッサ５００１はＤＳＰ５００２を突き放し制御方式で制御しており、プロセッサ５００１が記憶装置５００３から読み出したソフトウェアがＤＳＰ５００２で実行すべき処理の場合はプロセッサ５００１はＤＳＰ５００２へ処理要求を発行し、ＤＳＰ５００２は処理要求を受けた後、処理を実行する。外部メモリ５００５はプロセッサ５００１、ＤＳＰ５００２が処理を実行する際に使用する共有メモリである。調停回路５００８は、ＤＳＰ５００２で処理を実行中に新たな処理要求が発行された場合、すなわちＤＳＰ競合が発生した場合に、新たに発行された処理が実行できるか否かを判定する。また、使用状況監視装置５００９はＤＳＰ５００２の使用状況を監視し、要求に応じて使用状況を出力する。

【０１０１】

ここで、調停回路５００８の制御フローを図２２に示す。図２２において、５１０１はＤＳＰ５００２で実行する処理の処理要求が発生したか否かを判定する処理要求判定手段であり、５１０２は現在、ＤＳＰ５００２がプロセッサ５００１からの処理要求を待っている待ち状態か否かを判定するＤＳＰ状態判定手段であり、５１０３は処理要求に対しＤＳＰ５００２での処理の実行を促すＤＳＰ使用許可手段であり、５１０４は処理が終了したか否かを判定する処理終了判定手段である。ＤＳＰ５００２で処理を実行中に新たなＤＳＰ５００２への処理要求が発生した場合、つまり、ＤＳＰ競合が発生した場合、調停回路５００８は、一旦、処理要求を受理し、実際の処理に関しては先の処理が終了するまで待機（ｗａｉｔ）状態となる。ＤＳＰ５００２で実行する処理が頻繁に発生すると、ＤＳＰ競合が発生し、調停回路５００８により処理が待機（ｗａｉｔ）状態となる。その結果、ソフトウェア処理システム全体の処理速度が低下する。そこで第４の実施の形態では、ＤＳＰ競合を避け、処理速度の低下を防ぐ。

【０１０２】

図２３は、本発明の第４の実施の形態におけるソフトウェアのコンパイラの処理フロー図を示す。コンパイラは、まず、処理識別手段５２０１としてＤＳＰ５

００２で実行する処理を識別する。これは、処理識別手段５２０１がＤＳＰ５００２で実行する処理をあらかじめ認識していれば可能である。次に、識別した処理に対して、これと機能的には等価であるが、プロセッサ５００１で実行する処理である等価処理があらかじめライブラリ等で用意されているか否かを判定する。もし用意されていれば、コンパイラは処理識別手段５２０１で識別した処理に対して等価処理、記憶処理、使用状況判定処理、置換処理を追加する。ここで記憶処理とはＤＳＰ５００２で実行処理の処理要求が発行されてから処理が開始するまでの待ち時間を記録する処理であり、使用状況判定処理とは記憶処理で記憶した使用状況の情報からＤＳＰ競合が発生しているか否かを判定する処理であり、置換処理とは処理識別手段５２０１で識別した処理を等価処理へ置き換える処理を示す。

【０１０３】

図２４は、図２３の処理フローをもつコンパイラを使用したときのＤＳＰ５００２で実行する処理に対する変更を示す図である。図２４では、ＤＳＰ５００２で実行する処理を関数“ｄｓｐ_ｔａｓｋ（）”とし、この等価処理を“ｐｒｏｃ_ｄｓｐｔａｓｋ（）”としている。コンパイラは、あらかじめＤＳＰ５００２で実行する処理を登録することで処理識別手段５２０１を用いてソフトウェアの中から関数“ｄｓｐ_ｔａｓｋ（）”を識別する。次に、コンパイラは処理識別手段５２０１で識別した処理に対して上記の等価処理、記憶処理、使用状況判定処理、置換処理を付加する。

【０１０４】

まず、図２４（ｂ）の２行目では、ソフトウェアに使用状況判定処理を付加する。ここで“ｗａｉｔ_ｔｉｍｅ”はプロセッサ５００１がＤＳＰ５００２で実行する処理の処理要求を発行してから処理が開始するまでの待ち時間であり、“ＤＥＦＩＮＥＤ_ＴＩＭＥ”はあらかじめ設定された設定値である。待ち時間“ｗａｉｔ_ｔｉｍｅ”は、過去に発生したＤＳＰ５００２で実行する処理の待ち時間を表しており、現時刻に発生するＤＳＰ５００２で実行する処理要求を発行する前に“ｗａｉｔ_ｔｉｍｅ”を参照し、設定値である“ＤＥＦＩＮＥＤ_ＴＩＭＥ”と比較することでＤＳＰ競合を判定する。ＤＳＰ競合が発生中と判定され

た場合、図２２の制御フローに従う調停回路５００８では、ＤＳＰ競合の発生時に処理を待機状態とするため、そのときの待ち時間は長くなる。つまり、ＤＳＰ５００２で実行する処理の待ち時間がある程度大きくなるとＤＳＰ競合が発生したと推定できる。なお、設定値については、ソフトウェア処理システムの仕様に応じて的確な値を設定するのが望ましい。

【０１０５】

次に、図２４（ｂ）の３行目から６行目は、記憶処理を示す。使用状況判定処理によりＤＳＰ競合が発生していないと推定された場合、処理要求を発行した時点と“ｄｓｐ_ｔａｓｋ（）”の実行が開始される時点のタイマーの値を参照し、その差を算出することで待ち時間“ｗａｉｔ_ｔｉｍｅ”を算出している。

【０１０６】

図２４（ｂ）の９行目から１１行目は、置換処理を示す。使用状況判定処理によりＤＳＰ競合が発生していると推定された場合、さらに、ＤＳＰ競合を発生させることを防ぐため、プロセッサ５００１で実行する等価処理に置き換える。なお、置換処理で置換する処理は、処理の整合性を保つために機能的に等価であるが、ＤＳＰ５００２で実行しない処理が望ましい。

【０１０７】

第４の実施の形態では、上述のコンパイラが追加した等価処理、記憶処理、使用状況判定処理、置換処理により、使用状況監視処理の過程およびソフトウェア処理変更処理の過程を実現する。つまり、使用状況監視処理の過程とは、記憶処理により待ち時間“ｗａｉｔ_ｔｉｍｅ”を記憶しており、使用状況判定処理へ待ち時間“ｗａｉｔ_ｔｉｍｅ”を出力する。ソフトウェア処理変更処理の過程とは、識別した処理を等価処理へ置き換える置換処理であり、使用状況監視装置から出力される待ち時間に基づいてＤＳＰ競合を判定し、ＤＳＰ競合が発生していれば処理の置き換えを行う。

【０１０８】

ここで、本本実施の形態によるＤＳＰ競合の判定フローを図２５に示す。処理識別手段５２０１で識別した処理に対し、前回識別した処理を実行した際の待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”を比較する。

待ち時間“ｗａｉｔ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合には、使用状況判定処理はＤＳＰ競合が発生中であると判定し、等価処理への置き換えを行う。逆に待ち時間“ｗａｉｔ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”よりも小さい場合には、使用状況判定処理はＤＳＰ競合が発生していないと判定し、処理識別手段５２０１で識別した処理を待ち時間“ｗａｉｔ_ｔｉｍｅ”を記憶しながら実行する。

【０１０９】

ここで「前回識別した処理を実行した際の待ち時間」とは、２つ考えられる。１番目の待ち時間は処理識別手段２２０１で識別した処理の中で、現時刻から１回前に記録した待ち時間、つまり、記録した待ち時間の中で最も新しいものが該当する。２番目の待ち時間は、現時刻で識別した処理と同一の処理を過去に実行したときに記録した待ち時間が該当する。１番目の待ち時間では、待ち時間は常に最新のものを１つだけ記憶するが、２番目の待ち時間では、識別された処理毎に待ち時間を記憶する必要がある。

【０１１０】

以上、本実施の形態では、待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比較結果より、ＤＳＰ競合が発生中か否かを判定し、ＤＳＰ競合が発生中と判定された場合は、ＤＳＰ５００２で実行する処理をプロセッサ５００１で実行する処理に自動で置換する。その結果、動的にＤＳＰ競合を判定しながら自動で処理の割り当てを行うことができるとともに、新たなＤＳＰ競合を避けることができ、処理速度の低下を防止できる。なお、第４の実施の形態における上記の発明内容を後述の発明と区別するために第４の手法とする。

【０１１１】

さらに、第５の手法としての使用状況判定処理においては、使用状況監視装置から出力される競合情報である待ち時間“ｗａｉｔ_ｔｉｍｅ”と事前に設定しておいた設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”との比較結果に加えて、乱数を使用することでＤＳＰ競合の判定を行う。

【０１１２】

図２６は、待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭ

Ｅ”の比に対する等価処理への置き換えを行う確率を示している。待ち時間“ｗａｉｔ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”よりも小さい場合、言い換えれば、図２６で待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が１より小さい場合、使用状況判定処理は第４の手法と同様にＤＳＰ競合が発生していないと判定し、等価処理の置き換えを行わない。

【０１１３】

しかし、待ち時間“ｗａｉｔ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合、言い換えれば、図２６で待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が１以上の場合、図２６の確率に従って等価処理への置き換えを行ったり、あるいは等価処理への置き換えは行わずに待ち時間を再計測し記憶する。これは、処理時間“ｗａｉｔ_ｔｉｍｅ”が設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”以上の場合、第４の手法では必ずＤＳＰ競合が発生中と判定していたが、第５の手法では乱数を用いた確率に応じてＤＳＰ競合を判定するので、第４の手法とは逆の判定結果を出力する可能性を持たせた。逆の判定結果とは、ＤＳＰ競合が発生していないという判定結果である。

【０１１４】

そして、本実施の形態では図２５に示すＤＳＰ競合判定フローに従い、待ち時間を再度記録することになる。つまり、ＤＳＰ競合の判定に乱数を加えることで判定を見直しする。これにより、実際にはＤＳＰ競合が解消されたにもかかわらず、使用状況判定処理はＤＳＰ競合がそのまま継続されている、といった間違った判断を避けることができる。

【０１１５】

第５の手法は一連の処理の中でＤＳＰ競合が頻繁に発生するシステムに対して有効である。なお、図２６では、待ち時間“ｗａｉｔ_ｔｉｍｅ”と設定値“ＤＥＦＩＮＥＤ_ＴＩＭＥ”の比が大きければ大きいほど、つまり、待ち時間“ｗａｉｔ_ｔｉｍｅ”が設定値よりも大きな値をとればとるほど等価処理へ置き換える確率が小さくなっている。これは、待ち時間“ｗａｉｔ_ｔｉｍｅ”が長ければ長いほど近い未来にＤＳＰ競合は解消されるということが一般的に考えられるからである。

【０１１６】

なお、図２６の発生確率の値については、対象システムの仕様に応じて適当な値を与える必要がある。

【０１１７】

以上、第５の手法により使用状況判定処理におけるＤＳＰ競合の判定精度を向上できる。

【０１１８】

なお、本実施の形態の第５の手法では、ＤＳＰ競合の判定について乱数を用いた確率を使用したが、乱数を用いないＤＳＰ競合の判定も考えられる。例えば、ＤＳＰ競合時における待ち時間の平均値または最大値などの定数があらかじめ既知であれば、ＤＳＰ競合中と判定されてから一定周期ごとに待ち時間の再計測及び記録を行うことも可能である。

【０１１９】

次に、第６の手法について説明する。

【０１２０】

ＤＳＰ５００２で実行する処理が複数の箇所に存在する場合、その処理の呼び出し元（出現箇所）別にＤＳＰ競合を判定すれば判定精度は向上する。第６の方法では、処理識別手段５２０１で識別した処理の出現箇所毎に待ち時間を記憶し、それを使用することでＤＳＰ競合の判定を高精度に行う。

【０１２１】

図２１のソフトウェア処理システムでは、複数の処理が記述された関数を実行する際には、現在のプログラムカウンタを退避レジスタに一時的に退避させた後、関数の処理を実行し、関数の処理が終了した後、退避レジスタからプログラムカウンタを読み出してプログラムカウンタを元の値に戻す操作を行う。第６の手法では、出現箇所識別処理が上述の退避レジスタからプログラムカウンタを読み出すことで処理識別手段５２０１で識別した処理の出現箇所を特定する。

【０１２２】

図２５の判定フローをもつ第４の手法に、さらに出現箇所識別処理を追加した場合のＤＳＰ競合の判定フローを図２７に示す。

【０１２３】

まず、処理識別手段５２０１で識別された処理に対し、出現箇所識別処理により処理の出現箇所を調査する。上述したように退避レジスタを参照することで、出現箇所の特定は可能である。次に、出現箇所毎に記憶した待ち時間を参照する。次に、出現箇所毎に記憶した待ち時間を設定値と比較することでＤＳＰ競合が発生中か否かを判定し、ＤＳＰ競合が発生中であれば等価処理に置き換えて処理を実行し、ＤＳＰ競合が発生していなければ待ち時間を記憶しながら識別した処理をそのまま実行する。なお、出現箇所毎に記憶した待ち時間に待ち時間が記憶されていない場合は、ＤＳＰ競合が発生していないと判定されたとみなし、待ち時間を計測しながら処理識別手段５２０１で識別された処理を実行し、計測した待ち時間を新規に記憶する。

【０１２４】

第６の手法は、ループ処理など同一の処理が繰り返して処理される場合、出現箇所別に判定した待ち時間を参照することでＤＳＰ競合の判定精度が向上する。

【０１２５】

なお、本実施の形態における待ち時間“ｗａｉｔ_ｔｉｍｅ”を、第３の実施の形態の使用状況監視装置に記憶することでＤＳＰ競合の判定を行うことも可能である。このときのＤＳＰ競合の判定フローを図２８に示す。待ち時間を求めた後、第１の実施の形態における使用状況監視装置にも待ち時間を記憶することにより、ＤＳＰ競合を動的に判定でき、ＤＳＰ競合に伴う処理速度の低下を防ぐことが可能である。

【０１２６】

（第５の実施の形態）

図２９は第５の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。図２９において、ソフトウェア処理システムは、ＤＳＰ７００１とプロセッサ７００２と、ＤＳＰ使用状況監視装置７００３と、ＤＳＰ７００１とプロセッサ７００２で実行する処理である図３０のオブジェクトコードを格納したメモリ７００４で構成される。メモリ７００４は、ＤＳＰ７００１とプロセッサ７００２が処理を実行する際の共有メモリとして処理結

果や処理データを記憶する。

【０１２７】

図３０は第５の実施の形態にかかわるソフトウェア処理システムで実行するソフトウェアを示す。

【０１２８】

図３１は第５の実施の形態にかかわるＤＳＰ競合時の処理フローを示す。

【０１２９】

図３２は第５の実施の形態にかかわるソフトウェア処理システムにおける処理の流れを示す。

【０１３０】

第５の実施の形態によるソフトウェア処理システムを図２９～図３２を用いて具体的に説明する。

【０１３１】

図２９では、プロセッサ７００２はＤＳＰ７００１を突き放し制御方式で制御しており、プロセッサ７００２はメモリ７００４からソフトウェアをフェッチ、デコードし、ＤＳＰ７００１で実行する処理の場合、プロセッサ７００２はＤＳＰ７００１へ処理要求を発行することでＤＳＰ７００１は処理を実行する。ここで、プロセッサ７００２からＤＳＰ７００１への処理要求の発行について、さらに詳しく説明する。

【０１３２】

プロセッサ７００２は、まず、ＤＳＰ７００１にあるコマンドバッファに処理コマンドおよび処理を行うのに必要なデータを書き込む。データ書き込みの終了後、プロセッサ７００２はＤＳＰ７００１へ処理の開始要求を発行し、それを受けたＤＳＰ７００１は、コマンドバッファから処理コマンド、データを読み出すことで処理が実行される。つまり、第５の実施の形態において、プロセッサ７００２からＤＳＰ７００１への処理要求とは、ＤＳＰ７００１のコマンドバッファへの処理コマンド、データの書き込みおよび処理の開始要求の発行という一連の動作を意味する。

【０１３３】

また、図２９におけるＤＳＰ７００１の使用状況監視装置７００３は、ＤＳＰ７００１が現在処理を実行中か否かを監視するとともに、プロセッサ７００２がＤＳＰ７００１のコマンドバッファへ処理コマンドおよびデータの書き込みを行うか否かを監視する。そして、ＤＳＰ７００１が任意の処理を実行中にプロセッサ７００２が処理要求を発行するためにＤＳＰ７００１へのコマンドバッファへの書き込みを行った場合、ＤＳＰ使用状況監視装置７００３はプロセッサ７００２へ割り込み信号を与える。ＤＳＰ使用状況監視装置７００３からの割り込み信号を受けたプロセッサ７００２は、割り込みルーチンに分岐し、プロセッサ７００２からＤＳＰ７００１への処理の開始要求の発行を止める。その結果、ＤＳＰ７００１のコマンドバッファへは処理コマンド、データが格納されているが、ＤＳＰ７００１への処理の開始要求が止められるため、ＤＳＰ７００１は新たに処理を実行しない。つまり、割り込み信号により新規のＤＳＰ７００１の処理要求がキャンセルされる。

【０１３４】

さらに、割り込み信号を受けたプロセッサ７００２は、ＤＳＰ７００１で実行する処理の代わりに代替処理を行う。ここで代替処理とは、ＤＳＰ７００１を使用しない処理であり、本実施の形態ではＤＳＰ７００１で実行する予定だった処理と機能的には等価で、プロセッサ７００２上で実行する処理とする。

【０１３５】

以上の割り込み信号に対する動作を実現するために、本実施の形態のソフトウェア処理システムでは、ＤＳＰ７００１で実行する処理に対し、それと機能的に等価であるが、プロセッサ７００２で実行する処理をそれぞれ用意する。そして、ＤＳＰ使用状況監視装置７００３による割り込み信号を受けた場合、プロセッサ７００２はＤＳＰ７００１への処理要求を止め、等価な処理をプロセッサ７００２自身で代わりに実行する。

【０１３６】

図３０に本実施の形態のソフトウェア処理システムにおけるソフトウェアを示す（左の数字は行番号）。ｍａｉｎ関数の中にｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３，ｆｕｎｃ４，ｆｕｎｃ５，ｆｕｎｃ６，ｆｕｎｃ７，ｆｕｎｃ８があり、

その中でＤＳＰ７００１で実行する処理は、ｆｕｎｃ２，ｆｕｎｃ４，ｆｕｎｃ６である。ソフトウェアにはプロセッサ７００２上で実行する処理であるプロセッサ用ライブラリと、ＤＳＰ７００１上で実行する処理であるＤＳＰ用ライブラリが含まれており、図３０におけるｆｕｎｃ２，ｆｕｎｃ４，ｆｕｎｃ６については、それぞれｆｕｎｃ２_ｄｓｐ，ｆｕｎｃ４_ｄｓｐ，ｆｕｎｃ６_ｄｓｐというＤＳＰ用ライブラリを用いてＤＳＰ７００１上で処理を実行できるし、ｆｕｎｃ２_ｃｐｕ，ｆｕｎｃ４_ｃｐｕ，ｆｕｎｃ６_ｃｐｕというＣＰＵ用ライブラリを用いてプロセッサ７００２上でも実行可能とする。

【０１３７】

図３１は、図３０のソフトウェアを実行した際のＤＳＰ競合時の処理フローであり、先に説明したようにＤＳＰ７００１が他の処理を実行中にＤＳＰ７００１に対して新規に処理要求が発行されようとしている場合、プロセッサ７００２へ割り込み信号を与え、ＤＳＰ用ライブラリを使用する処理をＣＰＵ用ライブラリを使用する処理に置き換える。

【０１３８】

図３２は、本実施の形態のソフトウェア処理システムにおいて図３０のソフトウェアを実行したときの処理の流れを示している。

【０１３９】

図３２（ａ）はＤＳＰ７００１のタスクが競合しない場合を示す。プロセッサ７００２でのｆｕｎｃ１を実行する。プロセッサ７００２はｆｕｎｃ１を実行後、ＤＳＰ７００１に対しｆｕｎｃ２の処理要求を発行し、ＤＳＰ７００１ではｆｕｎｃ２_ｄｓｐを実行する。一方、プロセッサ７００２はｆｕｎｃ３を実行する。ＤＳＰ７００１はｆｕｎｃ２を実行後、処理結果をメモリ７００４に書き込む。なお、メモリ７００４はプロセッサ７００２、ＤＳＰ７００１からアクセス可能な共有メモリである。プロセッサ７００２でｆｕｎｃ３を実行後、ＤＳＰ７００１でｆｕｎｃ４の処理要求を発行し、ＤＳＰ７００１でｆｕｎｃ４_ｄｓｐを実行し、プロセッサ自身はｆｕｎｃ５を実行する。ＤＳＰ７００１はｆｕｎｃ４を実行後、処理結果をメモリ７００４に書き込む。プロセッサ７００２はｆｕｎｃ５を実行後、ＤＳＰ７００１でｆｕｎｃ６の処理要求を発行し、ＤＳＰ７０

０１でｆｕｎｃ６_ｄｓｐを実行し、プロセッサ自身はｆｕｎｃ７を実行する。ＤＳＰ７００１はｆｕｎｃ６を実行後、処理結果をメモリ７００４に書き込む。プロセッサ７００２は、以上の処理結果をメモリ７００４から読み取り、ｆｕｎｃ８を処理する。以上、図３２（ａ）ではＤＳＰ７００１のタスクが競合しないためプロセッサ７００２とＤＳＰ７００１の資源を効率良く使用できる。

【０１４０】

図３２（ｂ）では、ＤＳＰ７００１のタスクが競合する場合を示す。ここではｆｕｎｃ２_ｄｓｐが終了しないうちに、プロセッサ７００２からｆｕｎｃ４の処理要求を発行され、ＤＳＰ競合が起きている。その結果、ｆｕｎｃ４の処理はｆｕｎｃ２_ｄｓｐが終了するまで停止させられるのでシステムの処理速度が低下する。

【０１４１】

そこで本実施の形態では、ＤＳＰの使用状況監視装置７００３がＤＳＰ競合を判定し、ｆｕｎｃ４の処理要求を発行しようとするプロセッサ７００２へ割り込み信号を発生させる。それを受けたプロセッサ７００２は、自身が実行中のｆｕｎｃ５を中断し、ｆｕｎｃ４をＣＰＵ用ライブラリであるｆｕｎｃ４_ｃｐｕを用いてＣＰＵ上で処理し、メモリ７００４に処理結果を書き込む。その後、プロセッサ７００２は中断していたｆｕｎｃ５を再開する。

【０１４２】

このように、本実施の形態のソフトウェア処理システムでは、処理を実行しながらＤＳＰ競合を判定し、処理をＤＳＰ７００１かプロセッサ７００２に的確に割り振ることによって、ＤＳＰ競合を避け、処理速度の低下を防ぐ。さらに、ＤＳＰの使用状況監視装置７００３というハードウェアで処理することによって、割り込みルーチンを用意することを除いてソースコード４１０２に対する記述の修正、追加がないため、アセンブラやオブジェクトコードで提供された場合にも対応できる。

【０１４３】

（第６の実施の形態）

以下、本発明の第６の実施の形態を図面に基づいて説明する。

【０１４４】

図３３は、本発明の第６の実施の形態にかかわるソフトウェア処理システムのハードウェアの構成を示すブロック図である。

【０１４５】

図３３において、８００１はＣＰＵ、８００２はＤＳＰ、８００３はバス、８００４は外部メモリ、８００５は周辺回路、８００６はＣＰＵ８００１が持つメモリ、８００７はメモリ８００６中のバンク形式であるＤＳＰ８００２用のメモリバンク、８００８はメモリ８００６中のバンク形式であるＣＰＵ８００１用のメモリバンク、８００９はＤＳＰ８００２が処理を行っているかどうかを監視する使用状況監視装置、８０１０はＤＳＰ用メモリバンク８００７とＣＰＵ用メモリバンク８００８とを切り替えるバンク切り替え装置である。

【０１４６】

図３４は、本発明の第６の実施の形態にかかわるソフトウェア処理システムのソフトウェアの構成を示す図である。

【０１４７】

図３４において、８１０１はＣＰＵ８００１およびＤＳＰ８００２上で動作するプログラム、８１０２，８１０３，８１０４はプログラム８１０１中に記述され、ＤＳＰ８００２を使用するコマンドを含んでライブラリ化された処理、８１０５，８１０６，８１０７は処理８１０２，８１０３，８１０４とそれぞれ同等の機能を持つ、ＣＰＵ８００１を使用するコマンドを含んでライブラリ化された処理である。

【０１４８】

図３５は、本発明の第６の実施の形態にかかわるソフトウェア処理システムにおけるコンパイルフローを示すフローチャートである。

【０１４９】

図３５において、８２０１はプログラム８１０１をコンパイルするコンパイル手段、８２０２はＤＳＰ８００２で行われる処理を識別する処理識別手段、８２０３はＤＳＰ用メモリバンク８００７またはＣＰＵ用メモリバンク８００８にマッピングするマッピング手段である。

【０１５０】

以上のように構成した本実施の形態の動作について説明する。

【０１５１】

なお、図３６は、本発明の第６の実施の形態にかかわるソフトウェア処理システムにおけるメモリバンクへのソフトウェアのマッピングを示す図である。

【０１５２】

本実施の形態において、ＤＳＰ８００２はＣＰＵ８００１からの処理要求を受付けることにより処理を実行し、ＣＰＵ８００１とＤＳＰ８００２との間の制御方式は突き放し制御方式である。プログラム８１０１に記述される処理ｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３は、ＤＳＰ８００２またはＣＰＵ８００１を使用するコマンドを含み、手順として処理ｆｕｎｃ１、処理ｆｕｎｃ２、処理ｆｕｎｃ３の順番であって、それぞれの処理にはデータ依存がないものとする。

【０１５３】

プログラム８１０１が、コンパイラが有するコンパイル手段８２０１によってコンパイルされる際、処理識別手段８２０２によってＤＳＰ８００２を使用するコマンドを含んだ処理である処理ｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３を識別し、図３６に示すように、メモリバンクのマッピング手段８２０３によって、ＤＳＰ用メモリバンク８００７にはＤＳＰ８００２用のライブラリ化された処理８１０２，８１０３，８１０４がマッピングされる。同様に、ＣＰＵ用メモリバンク８００８にはＣＰＵ８００１用のライブラリ化された処理８１０５，８１０６，８１０７がマッピングされる。このとき、処理８１０２と処理８１０５は同じアドレス番地を開始アドレスとする。処理８１０３と処理８１０６、処理８１０４と処理８１０７についても同様である。このようにコンパイラは同じ名前である別々のプロセッサ用の処理を認識してコンパイルすることが可能であるという特徴を持つ。

【０１５４】

プログラム８１０１が処理される際、最初に処理ｆｕｎｃ１が行われるとする。処理ｆｕｎｃ１が呼び出され、処理ｆｕｎｃ１がマッピングされている開始アドレスの命令がＣＰＵ８００１によってフェッチされた瞬間に、バンク切り替え

装置８０１０によってバンク切り替えのロックが実行される。このロックの実行によって、他の処理によるバンク切り替えが実行できないようになる。また、ＤＳＰ８００２が処理を行っているかどうかの情報が使用状況監視装置８００９によって識別されていて、その情報はバンク切り替え装置８０１０への入力信号となり、バンク切り替え装置８０１０によってメモリ８００６へバンク切り替えを実行するための要求が行われている。

【０１５５】

まず、ＤＳＰ８００２が他の処理を行っていない場合、バンク切り替えは、表側がＤＳＰ用メモリバンク８００７であり、裏側がＣＰＵ用メモリバンク８００８である。先ほどのバンク切り替えのロックによって、このバンク切り替えの状態は維持される。バンク切り替えのロックが実行されている間は、バンク切り替えの表側がＤＳＰ用メモリバンク８００７になっているので、ＣＰＵ８００１はＤＳＰ用メモリバンク８００７から処理８１０２を取り出す。処理８１０２にはＤＳＰ８００２を使用するコマンドが含まれていて、ＣＰＵ８００１によって、このＤＳＰ８００２を使用するコマンドが発行され、ＤＳＰ８００２によって処理が行われる。ＤＳＰ８００２は突き放し制御であるので、このコマンドの処理が終了する時間を特定することはできない。処理ｆｕｎｃ１の呼び出し元であるプログラム８１０１のｍａｉｎ関数へ戻る際、このｍａｉｎ関数へ戻る命令がＣＰＵ８００１によってフェッチされた瞬間に、バンク切り替え装置８０１０によってバンク切り替えのアンロックが実行される。そして、メモリ８００６へのバンク切り替え要求が受け付けられ、結果として、表側がＣＰＵ用メモリバンク８００８となり、裏側がＤＳＰ用メモリバンク８００７となる。

【０１５６】

これとは逆の場合、つまり、処理ｆｕｎｃ１が呼び出された際、ＤＳＰ８００２が他の処理を行っていた場合、バンク切り替えは、表側がＣＰＵ用メモリバンク８００８であり、裏側がＤＳＰ用メモリバンク８００７である。バンク切り替えのロックが実行されているので、このバンク切り替えの状態はアンロックが実行されるまで維持される。ＤＳＰ８００２が他の処理のよって使用されているので、バンク切り替え装置８０１０はメモリ８００６へ、バンク切り替えを表側が

ＣＰＵ用メモリバンク８００８、裏側がＤＳＰ用メモリバンク８００７となるように要求することになる。

【０１５７】

バンク切り替えは表側がＣＰＵ用メモリバンク８００８となっているので、ＣＰＵ８００１はＣＰＵ用メモリバンク８００８から処理８１０５を取り出す。処理８１０５は、ＤＳＰ８００２を使用するコマンドを含んだ処理８１０２と同等の機能を持っていて、ＣＰＵ８００１を使用するコマンドを含んでいるので、ＣＰＵ８００１によって処理が行われる。処理８１０５の処理が終了し、処理ｆｕｎｃ１の呼び出し元であるプログラム８１０１のｍａｉｎ関数へ戻る際、このｍａｉｎ関数へ戻る命令がＣＰＵ８００１によってフェッチされた瞬間に、バンク切り替え装置８０１０によってバンク切り替えのアンロックが実行される。そして、メモリ８００６へのバンク切り替え要求が受け付けられ、結果として、表側がＤＳＰ用メモリバンク８００７となり、裏側がＣＰＵ用メモリバンク８００８となる。ＣＰＵ８００１が処理８１０５を行っている間に、ＤＳＰ８００２を使用していた他の処理が終了した場合は、使用状況監視装置８００９の出力を受けて、バンク切り替え装置８０１０が、バンク切り替えの表側をＤＳＰ用メモリバンク８００７とし、裏側をＣＰＵ用メモリバンク８００８とする要求を行うことになる。しかし、この要求が受け付けられてバンク切り替えが実行されるのは、前述と同様にバンク切り替えのアンロックが実行されてからである。

【０１５８】

次に、処理ｆｕｎｃ１のＤＳＰ８００２を使用する処理８１０２中のコマンドが、ＤＳＰ８００２によって処理が行われている間、次の処理である処理ｆｕｎｃ２が呼び出されたとする。このとき、ＤＳＰ８００２は使用中であり、先ほどのバンク切り替えのアンロックはバンク切り替え装置８０１０によって実行されているので、バンク切り替えは表側がＣＰＵ用メモリバンク８００８となり、裏側がＤＳＰ用メモリバンク８００７となっている。処理ｆｕｎｃ２が呼び出され、処理ｆｕｎｃ２がマッピングされている開始アドレスの命令がＣＰＵ８００１によってフェッチされた瞬間に、バンク切り替え装置８０１０によってバンク切り替えのロックが実行される。このロックの実行によって、他の処理によるバン

ク切り替えが実行できないようになる。

【０１５９】

バンク切り替えは表側がＣＰＵ用メモリバンク８００８となっているので、ＣＰＵ８００１はＣＰＵ用メモリバンク８００８から処理８１０６を取り出す。処理８１０６は、ＤＳＰ８００２を使用するコマンドを含んだ処理８１０３と同等の機能を持っていて、ＣＰＵ８００１を使用するコマンドを含んでいるので、ＣＰＵ８００１によって処理が行われる。処理８１０６の処理が終了し、処理ｆｕｎｃ２の呼び出し元であるプログラム８１０１のｍａｉｎ関数へ戻る際、このｍａｉｎ関数へ戻る命令がＣＰＵ８００１によってフェッチされた瞬間に、バンク切り替え装置８０１０によってバンク切り替えのアンロックが実行される。そして、メモリ８００６へのバンク切り替え要求が受け付けられ、結果として、表側がＤＳＰ用メモリバンク８００７となり、裏側がＣＰＵ用メモリバンク８００８となる。ＣＰＵ８００１が処理８１０６を行っている間に、ＤＳＰ８００２を使用していた処理８１０２中のコマンドが終了した場合は、先ほどの処理ｆｕｎｃ１の処理を行う際にＤＳＰ８００２が使用中であった場合と同様に、バンク切り替え装置８０１０によってメモリ８００６へバンク切り替えの要求が行われ、バンク切り替えのアンロックが実行されてから、この要求が受け付けられることになる。

【０１６０】

最後に処理ｆｕｎｃ３が呼び出される。先ほどのＤＳＰ８００２が使用中でない場合の処理ｆｕｎｃ１の場合と同様に、バンク切り替え装置８０１０によって、バンク切り替えのロック、バンク切り替え要求が行われ、ＤＳＰ用メモリバンク８００７からＤＳＰ８００２を使用するコマンドを含んだ処理８１０４がＣＰＵ８００１によって取り出され、ＤＳＰ８００２を使用するコマンドは突き放し処理としてＤＳＰ８００２によって実行される。プログラム８１０１のｍａｉｎ関数へ戻る際のバンク切り替えのアンロックの実行、バンク切り替えの表側をＣＰＵ用メモリバンク８００８とし、裏側をＤＳＰ用メモリバンク８００７とするバンク切り替えの実行についても同様である。

【０１６１】

以上のように本実施の形態によれば、バンク切り替え装置８０１０を用いた方法によって、ＤＳＰ８００２におけるＤＳＰ競合の発生を軽減し、ＤＳＰ８００２を効率良く使用することが可能となる。この方法を実現するために、プログラム８１０１を修正する必要がなくそのまま適用できるので、ソフトウェアのオーバーヘッドがほとんどない。また、使用状況監視装置８００９およびバンク切り替え装置８０１０という専用のハードウェアを用いることによって、バス８００３の使用状況の監視およびソフトウェア処理変更を比較的高速に実現することができる。

【０１６２】

なお、本実施の形態では、プロセッサを１つのＣＰＵ、リソースを１つのＤＳＰとしたが、それぞれ複数のＣＰＵ、ＤＳＰでも良く、少なくとも１つのリソースとそのリソースを使用する少なくとも１つのプロセッサであれば良いものである。

【０１６３】

【発明の効果】

以上説明したように、本発明によれば、リソースの競合を動的に判定し、リソースを使用する処理の処理方法を変更することでリソース競合時の処理の停止を避け、システムの処理速度の低下を抑制することができる。

【０１６４】

その第１の方法としては、ソフトウェアをコンパイルする段階で、リソースを使用する処理を識別し、その処理に対しリソース競合を判定する処理と、その判定結果に基づいて処理を別の等価処理に置き換える一連の処理を組み込む。これにより、リソース競合を避け、システムの処理速度の低下を避けることができる。

【０１６５】

さらに、リソースが共有メモリに接続されたバスの場合、バス競合の履歴を採取することで、さらに高精度なリソース競合の判定が可能である。

【０１６６】

また、第２の方法としては、プロセッサで任意の処理を実行中に新規の処理要

求が発生した場合、割り込み信号をプロセッサへ与えることで新規の処理を別プロセッサで代用させる。本発明を実現するために割り込みルーチンを用意する以外は、もともとのソフトウェアに対して処理の変更、追加をする必要が無く、ソフトウェアをそのまま適用でき、さらにソフトウェアによる競合対策の処理がないため、ソフトウェアのオーバーヘッドが少ない。

【０１６７】

また、第３の方法として、あらかじめ同一アドレスを割り当てた複数のメモリバンクを用意し、プロセッサが任意の処理を実行中に新規の処理要求が発生した場合、メモリバンクを切り替えることで新規処理を別のプロセッサで代わりに実行させる。これについては、競合対策の処理がメモリバンク切り替え回路のみで実現できるため、他の方法に比べて高速である。

【０１６８】

また、第４の方法として、処理の待ち時間や処理時間などを計測することでリソース競合が発生中か否かを判定し、リソース競合が発生中ならば新たにリソース競合を引き起こす処理を発行しない。これにより、競合によるシステムの処理速度の低下を防ぐことが可能である。

【０１６９】

さらに、第４の方法に対し、定期的または不定期に競合判定を補正することで、さらに高精度な競合判定が可能である。

【０１７０】

さらに、第４の方法に対し、競合情報を競合を引き起こす処理の出現箇所とともに記憶することで、ループ処理における競合判定を高精度に行うことが可能である。

【図面の簡単な説明】

【図１】　本発明の実施の形態におけるソフトウェア処理システムの基本構成を示すブロック図

【図２】　本発明の第１の実施の形態におけるソフトウェア処理システムのハードウェアの構成を示すブロック図

【図３】　本発明の第１の実施の形態におけるソフトウェア処理システムのバ

スの使用状況監視装置の構成を示す図

【図４】　本発明の第１の実施の形態におけるソフトウェア処理システムのソフトウェアの構成を示す図

【図５】　本発明の第１の実施の形態におけるソフトウェア処理システムにおけるコンパイルフローを示すフローチャート

【図６】　本発明の第１の実施の形態におけるソフトウェア処理システムにおけるバスアクセス識別装置が識別するバスアクセスの状態を示す図

【図７】　本発明の第１の実施の形態におけるソフトウェア処理システムにおける処理付加手段によるソフトウェアの変更を示す図

【図８】　本発明の第２の実施の形態におけるソフトウェア処理システムのハードウェアの構成を示すブロック図

【図９】　本発明の第２の実施の形態におけるソフトウェア処理システムのバス調停回路の制御フローを示すフローチャート

【図１０】　本発明の第２の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図１１】　本発明の第２の実施の形態におけるソフトウェア処理システムの外部メモリへ頻繁にアクセスする処理に対する変更を示す図

【図１２】　本発明の第２の実施の形態におけるソフトウェア処理システムのバス競合の判定フローを示すフローチャート

【図１３】　本発明の第２の実施の形態におけるソフトウェア処理システムの等価処理への置換を行う確率を示す図

【図１４】　本発明の第２の実施の形態におけるソフトウェア処理システムの出現箇所を考慮した場合のバス競合の判定フローを示すフローチャート

【図１５】　本発明の第２の実施の形態におけるソフトウェア処理システムのバス競合の判定フローを示すフローチャート

【図１６】　本発明の第３の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図１７】　本発明の第３の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図１８】　本発明の第３の実施の形態におけるソフトウェア処理システムのコンパイラによる処理の変更を示す図

【図１９】　本発明の第３の実施の形態におけるソフトウェア処理システムのＤＳＰ競合の判定フローを示すフローチャート

【図２０】　本発明の第３の実施の形態におけるソフトウェア処理システムの処理の流れを示すシーケンス図

【図２１】　本発明の第４の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図２２】　本発明の第４の実施の形態におけるソフトウェア処理システムの調停回路の制御フローを示すフローチャート

【図２３】　本発明の第４の実施の形態におけるソフトウェア処理システムのコンパイラの処理フローを示すフローチャート

【図２４】　本発明の第４の実施の形態におけるソフトウェア処理システムのＤＳＰで実行する処理に対する変更を示す図

【図２５】　本発明の第４の実施の形態におけるソフトウェア処理システムのＤＳＰ競合の判定フローを示すフローチャート

【図２６】　本発明の第４の実施の形態におけるソフトウェア処理システムの等価処理への置換を行う確率を示す図

【図２７】　本発明の第４の実施の形態におけるソフトウェア処理システムの出現箇所を考慮した場合のＤＳＰ競合の判定フローを示すフローチャート

【図２８】　本発明の第４の実施の形態におけるソフトウェア処理システムのＤＳＰ競合の判定フローを示すフローチャート

【図２９】　本発明の第５の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図３０】　本発明の第５の実施の形態におけるソフトウェア処理システムのソフトウェアを示す図

【図３１】　本発明の第５の実施の形態におけるソフトウェア処理システムのＤＳＰ競合時の処理フローを示すフローチャート

【図３２】　本発明の第５の実施の形態におけるソフトウェア処理システムに

おける処理の流れを示すシーケンス図

【図３３】　本発明の第６の実施の形態におけるソフトウェア処理システムのハードウェア構成を示すブロック図

【図３４】　本発明の第６の実施の形態におけるソフトウェア処理システムのソフトウェアの構成を示す図

【図３５】　本発明の第６の実施の形態におけるソフトウェア処理システムのコンパイルフローを示すフローチャート

【図３６】　本発明の第６の実施の形態におけるソフトウェア処理システムのメモリバンクマッピングを示す図

【符号の説明】

１０１　プロセッサ

１０２　リソース

１０３　使用状況監視処理手段

１０４　ソフトウェア処理変更処理手段

１００１　ＣＰＵ

１００２　バス

１００３　周辺回路

１００４　外部メモリ

１００５　使用状況監視処理装置

１１０１　バスアクセス識別装置

１１０２　バスアクセス状態レジスタ

１２０１　プログラム

１３０１　コンパイル手段

１３０２　処理識別手段

１３０３　処理付加手段

２００１　プロセッサ

２００２　ＤＳＰ

２００３　記憶装置

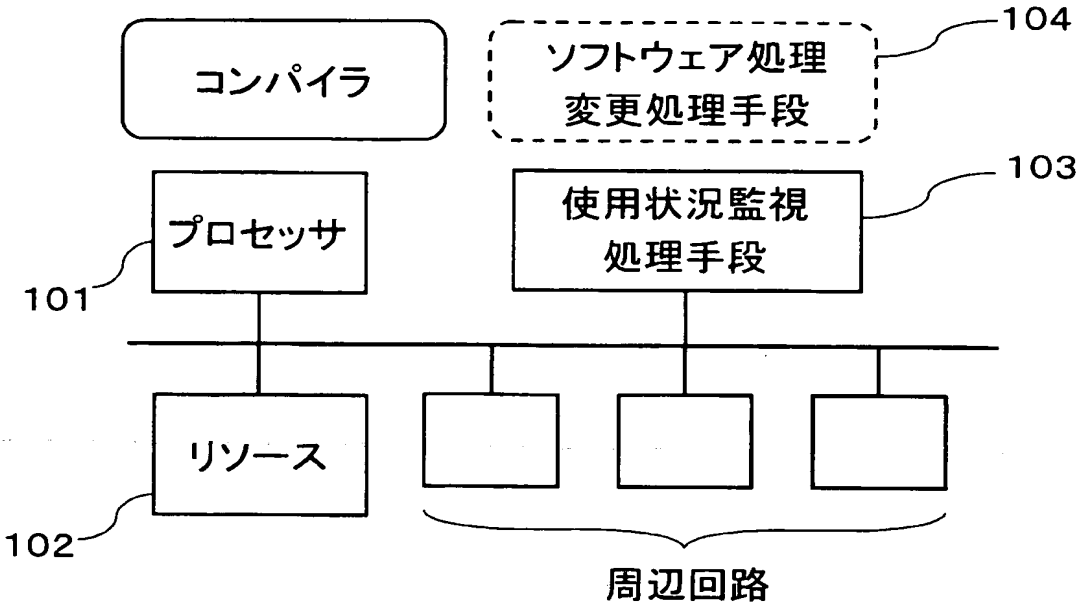
２００４　周辺回路

２００５　バス調停回路
２００６　外部メモリ
２００７　共有バス
２１０１　アクセス要求判定手段
２１０２　メモリ状態判定手段
２１０３　メモリアクセス許可手段
２１０４　処理終了判定手段
２２０１　処理識別手段
４００１　ＤＳＰ
４００２　プロセッサ
４００３　ＤＳＰ使用状況監視装置
４００４　メモリ
４１０１　コンパイラ
４１０２　ソースコード
４１０３　プロセッサ用ライブラリ
４１０４　ＤＳＰ用ライブラリ
４１０５　オブジェクトコード
４１０６　プロセッサ用ライブラリ
４１０７　ＤＳＰ用ライブラリ
５００１　プロセッサ
５００２　ＤＳＰ
５００３　記憶装置
５００４　周辺回路
５００５　外部メモリ
５００６　共有バス
５００７　バス調停回路
５００８　調停回路
５１０１　処理要求判定手段
５１０２　ＤＳＰ状態判定手段

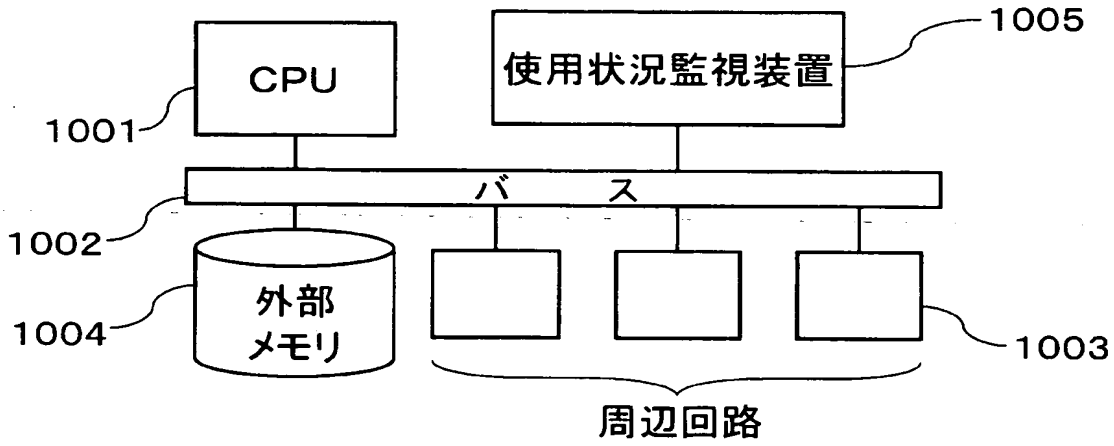
５１０３　ＤＳＰ使用許可手段
５１０４　処理終了判定手段
５２０１　処理識別手段
７００１　ＤＳＰ
７００２　プロセッサ
７００３　ＤＳＰ使用状況監視装置
７００４　メモリ
８００１　ＣＰＵ
８００２　ＤＳＰ
８００３　バス
８００４　外部メモリ
８００５　周辺回路
８００６　メモリ
８００７　ＤＳＰ用メモリバンク
８００８　ＣＰＵ用メモリバンク
８００９　使用状況監視装置
８０１０　バンク切り替え装置
８１０１　プログラム
８２０１　コンパイル手段
８２０２　処理識別手段
８２０３　マッピング手段

【書類名】　図面

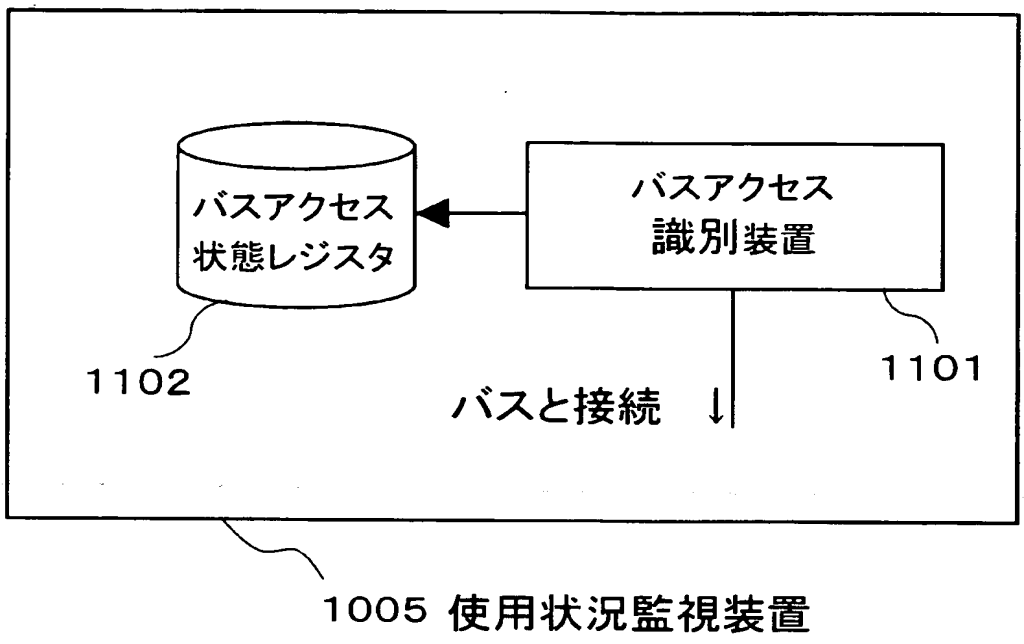
【図１】


コンパイラ
ソフトウェア処理
変更処理手段
104
プロセッサ
101
使用状況監視
処理手段
103
リソース
102
周辺回路

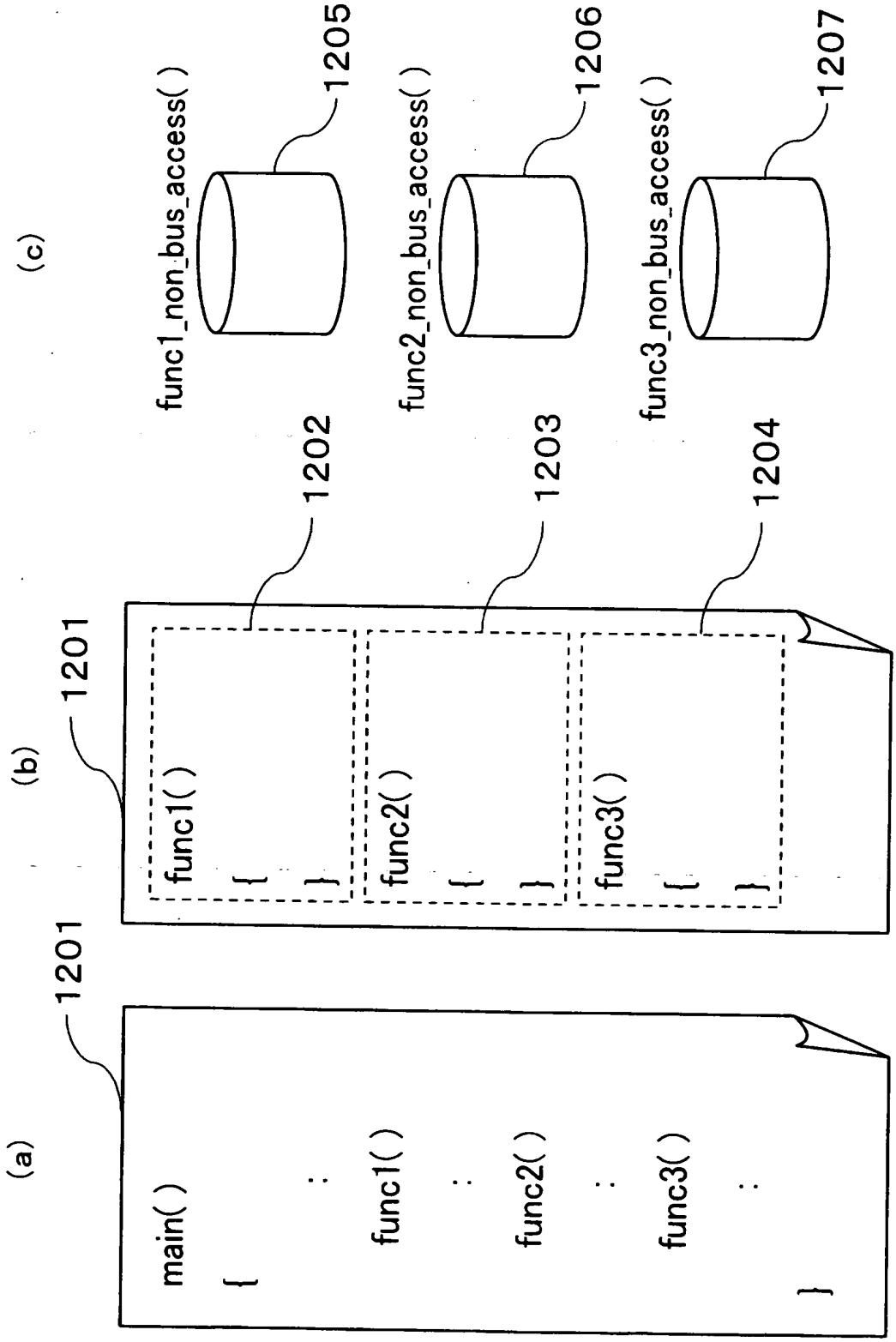
【図２】


CPU
1001
使用状況監視装置
1005
バス
1002
外部
メモリ
1004
1003
周辺回路

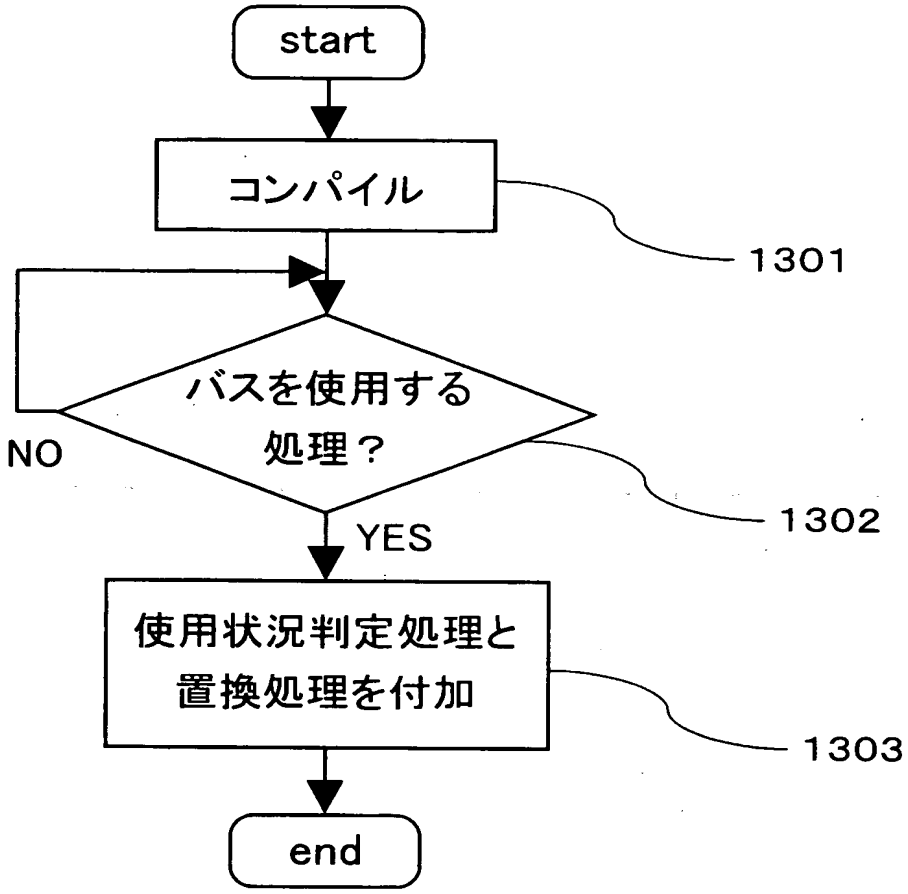
【図３】



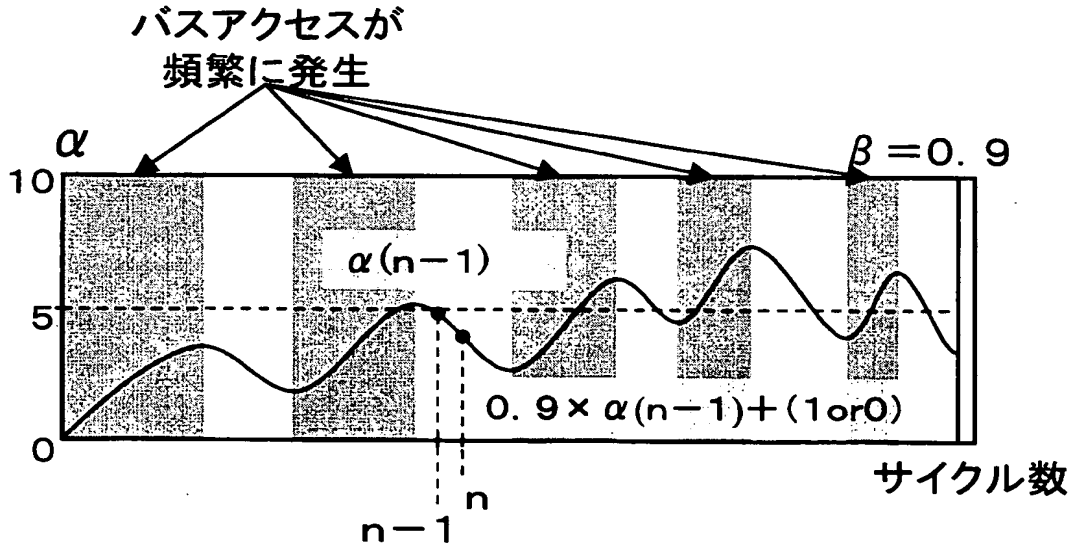
【図４】


(a)
1201
main( )
{
func1( )
func2( )
func3( )
}
(b)
1201
1202
1203
1204
func1( )
{
}
func2( )
{
}
func3( )
{
}
(c)
1205
1206
1207
func1_non_bus_access( )
func2_non_bus_access( )
func3_non_bus_access( )

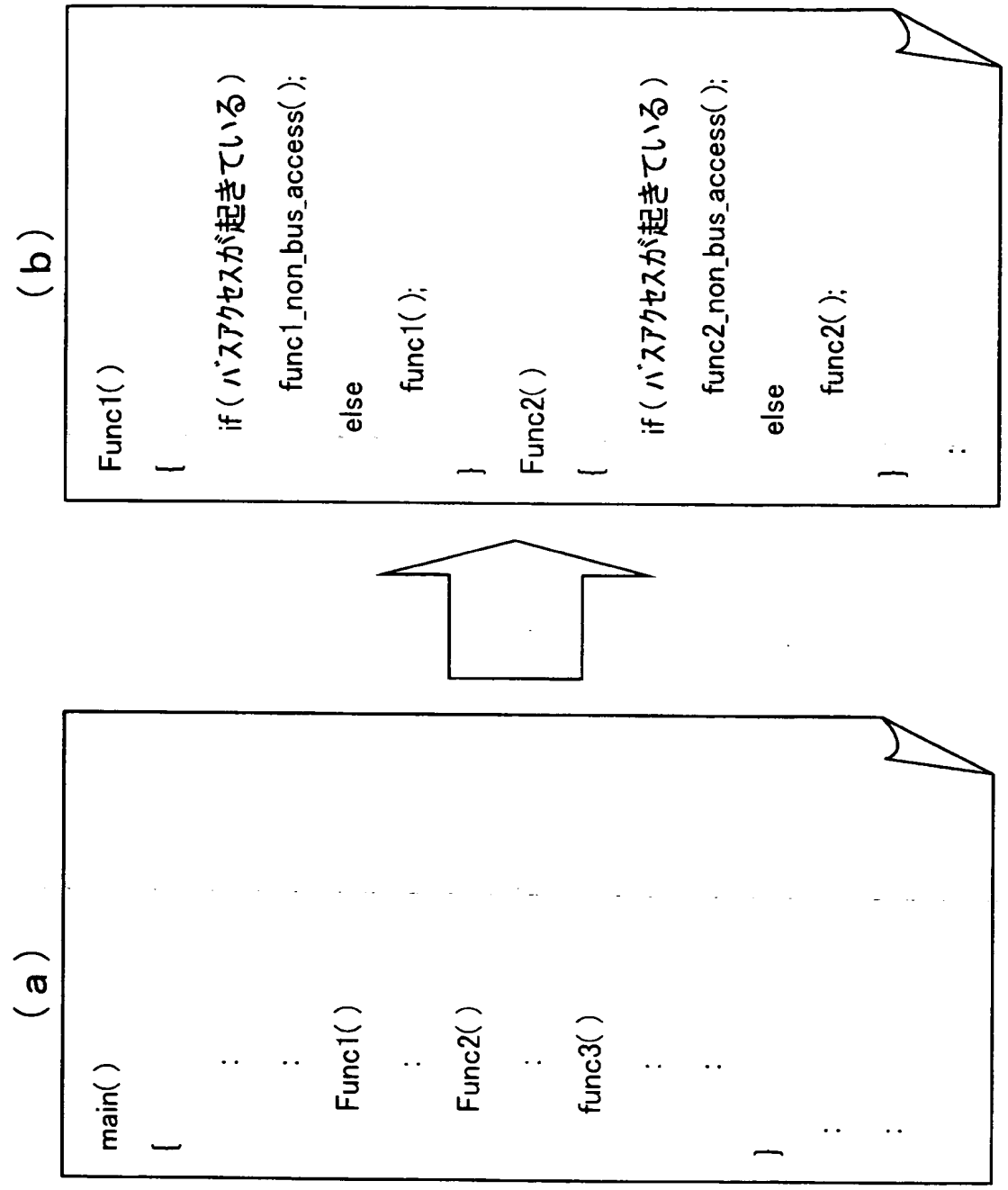
【図５】


start
コンパイル
1301
バスを使用する
処理？
NO
YES
1302
使用状況判定処理と
置換処理を付加
1303
end

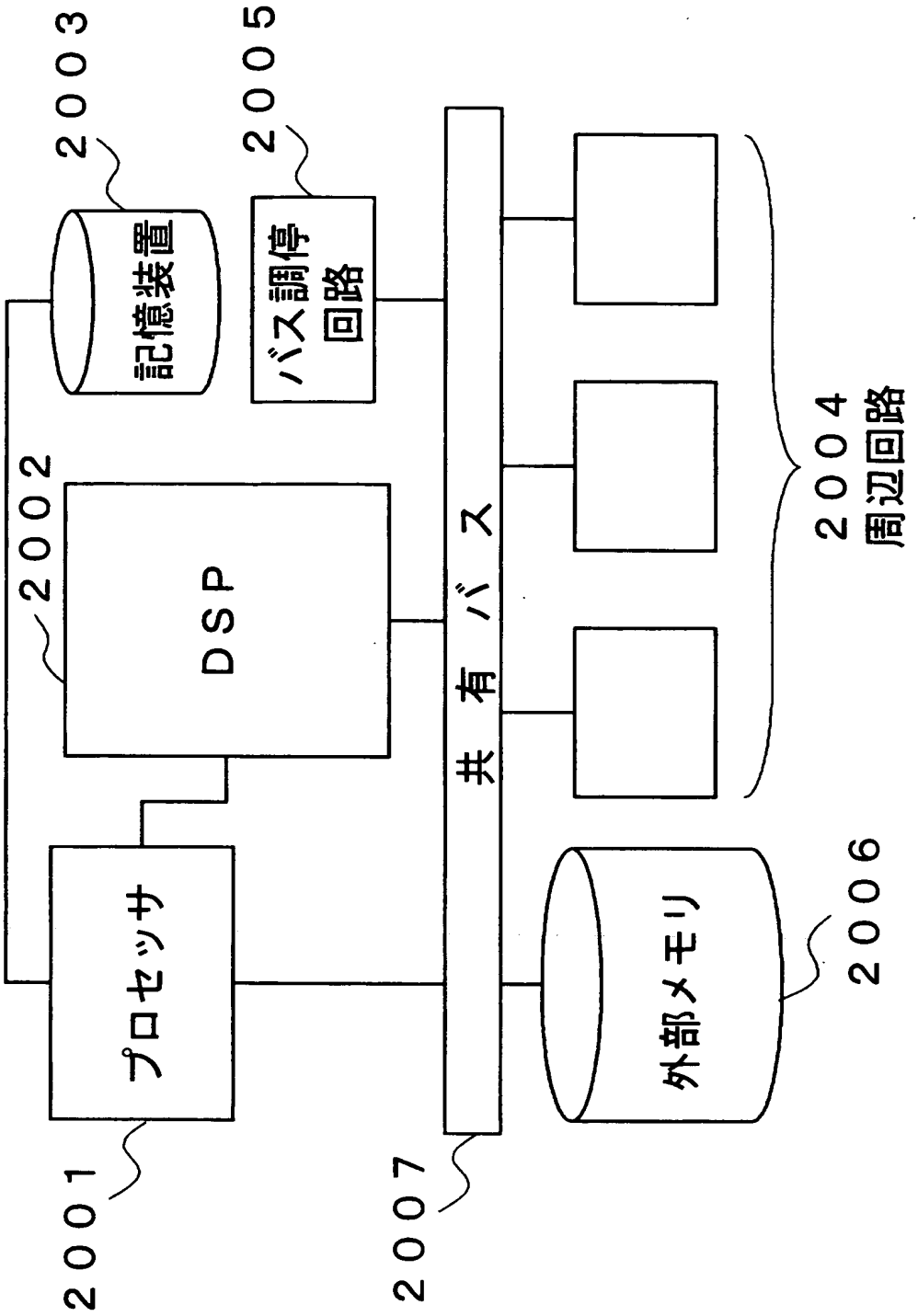
【図６】


バスアクセスが
頻繁に発生
α
β＝0．9
10
5
0
α(n－1)
0．9×α(n－1)＋(1or0)
サイクル数
n
n－1

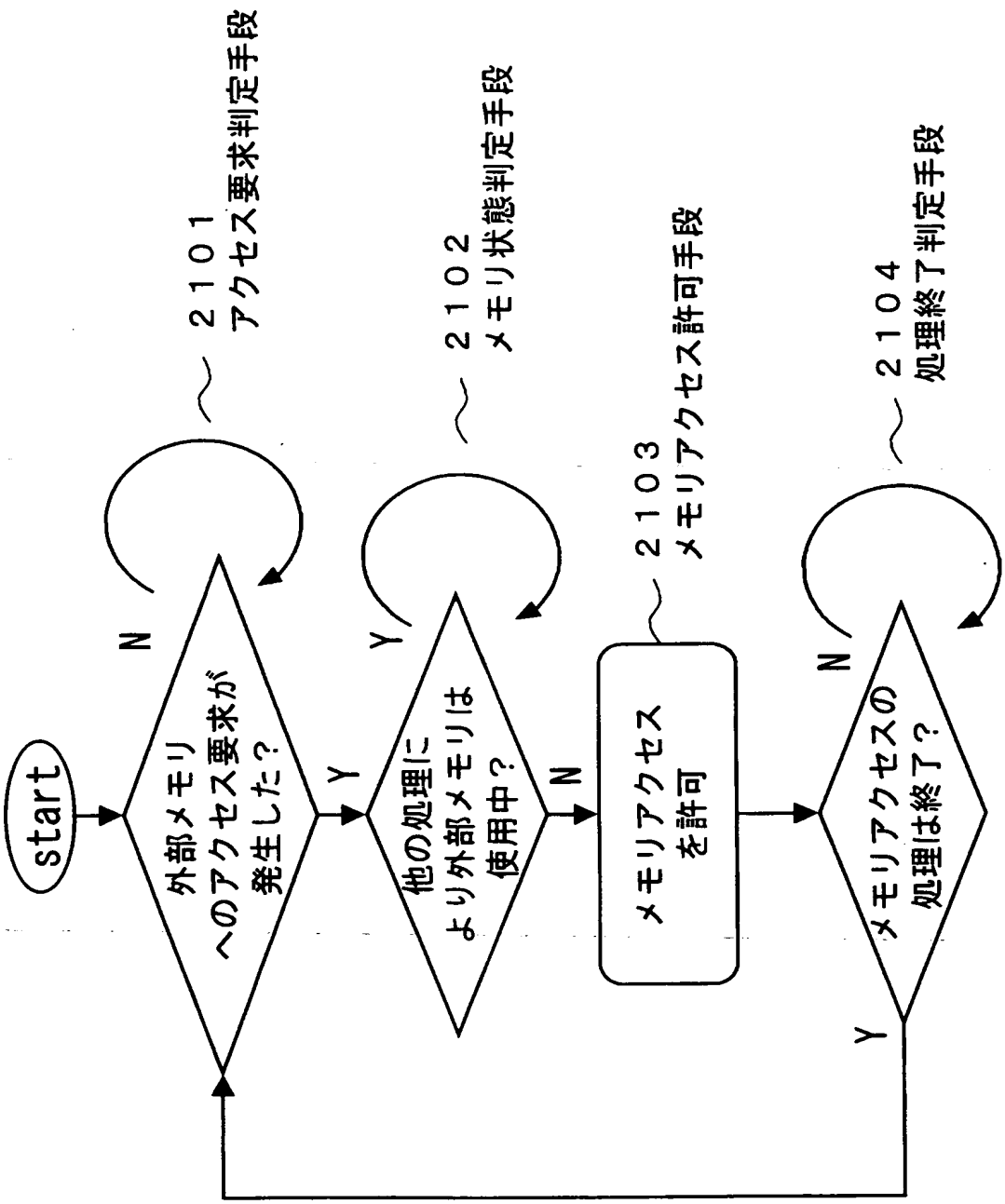
【図７】


(a)
main( )
{
Func1( )
Func2( )
func3( )
}
(b)
Func1( )
{
if (バスアクセスが起きている)
func1_non_bus_access( );
else
func1( );
}
Func2( )
{
if (バスアクセスが起きている)
func2_non_bus_access( );
else
func2( );
}

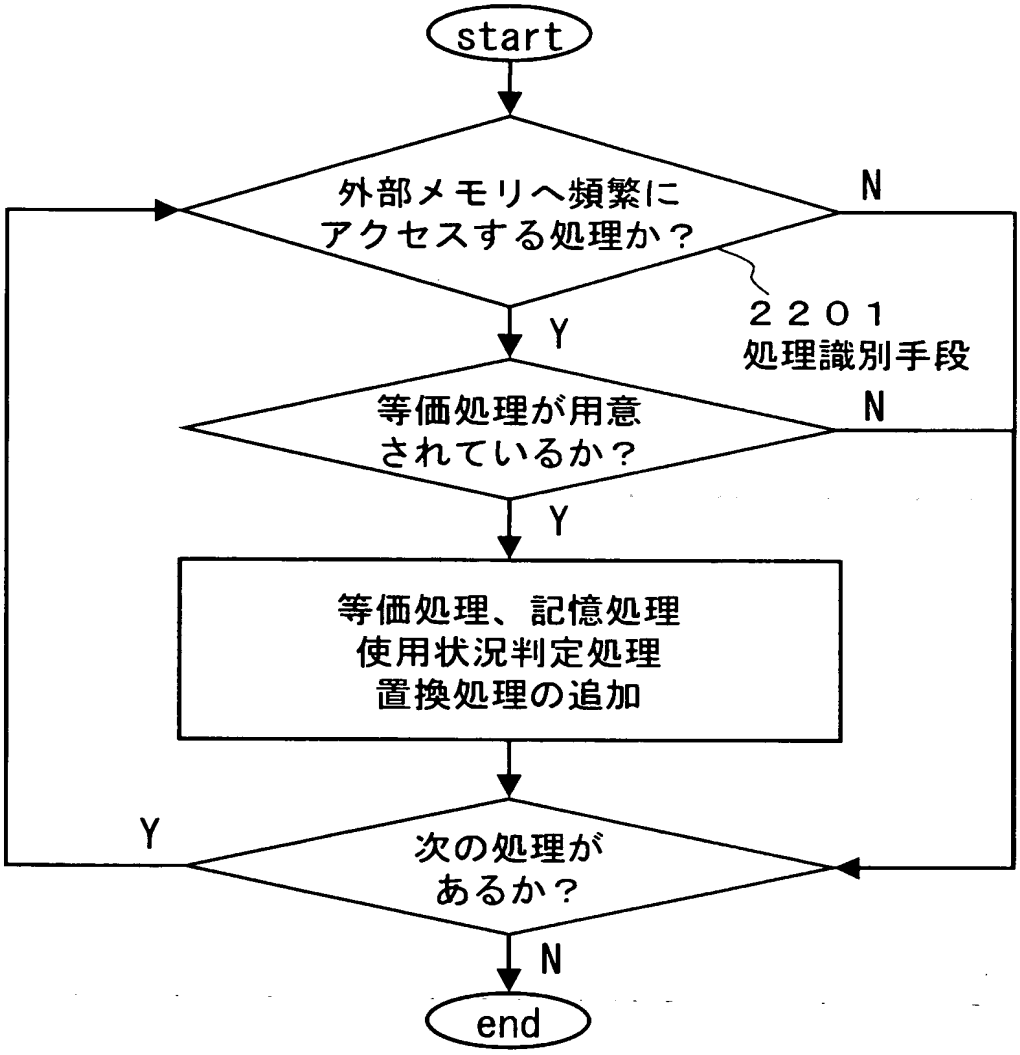
【図８】


2003
記憶装置
2005
バス調停
回路
2002
DSP
共有バス
2004
周辺回路
プロセッサ
外部メモリ
2006
2001
2007

【図９】


start
外部メモリへのアクセス要求が発生した？
N
Y
２１０１
アクセス要求判定手段
他の処理により外部メモリは使用中？
Y
N
２１０２
メモリ状態判定手段
メモリアクセスを許可
２１０３
メモリアクセス許可手段
メモリアクセスの処理は終了？
N
Y
２１０４
処理終了判定手段

【図１０】


start
外部メモリへ頻繁に
アクセスする処理か？
N
２２０１
処理識別手段
Y
等価処理が用意
されているか？
N
Y
等価処理、記憶処理
使用状況判定処理
置換処理の追加
Y
次の処理が
あるか？
N
end

【図１１】

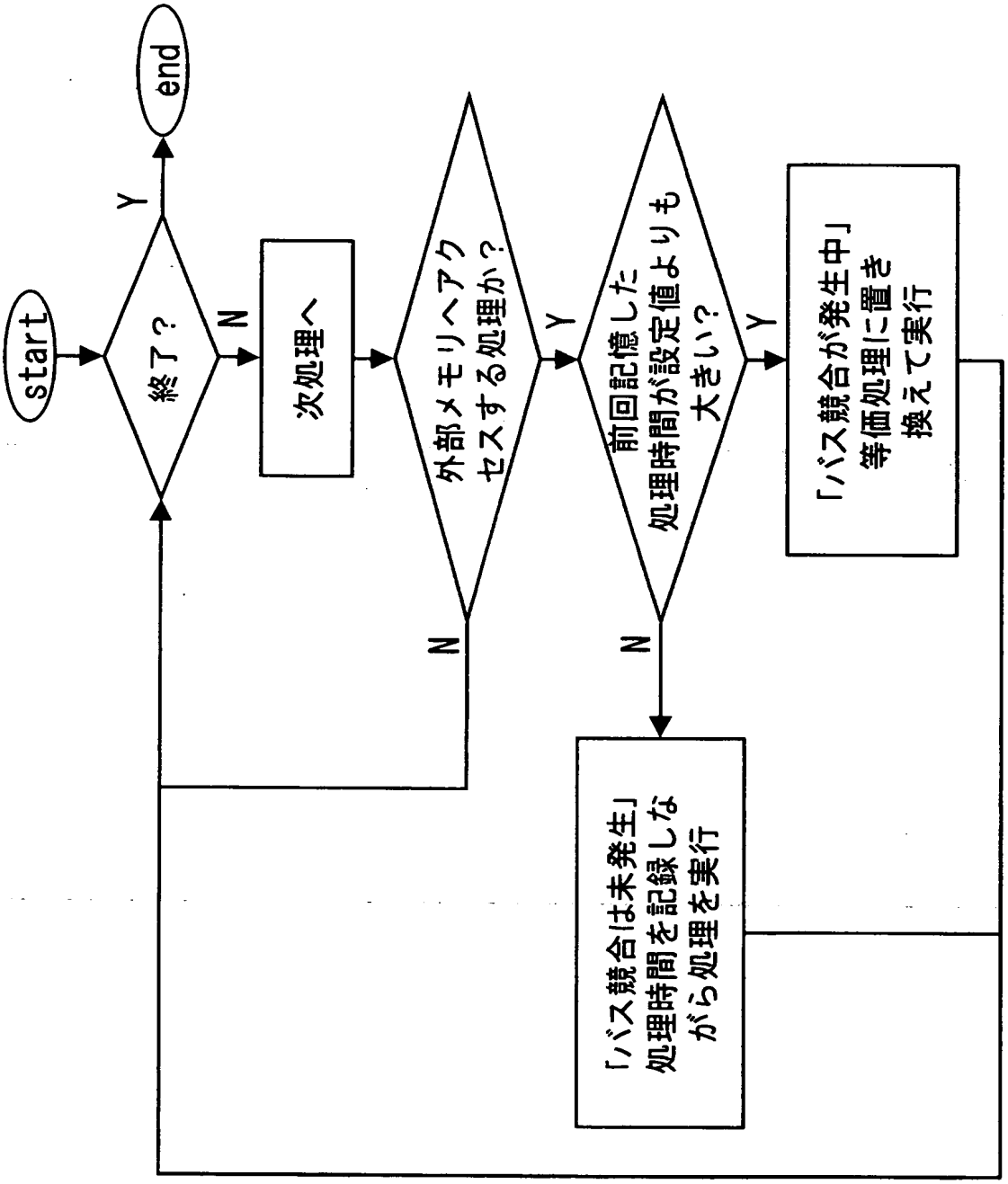
```
1: proc_task1() {
2:     mem_acs_many();
3: };
```

(a)コンパイラ実行前

```
1: proc_task1() {
2:     if( task_time < DEFINED_TIME ) {
3:         start_time = timer_count;
4:         mem_acs_many();
5:         end_time   = timer_count;
6:         task_time = end_time - start_time;
7:     }
8:     else{
9:         mem_acs_few();
10:    };
11: };
```

(b)コンパイラ実行後

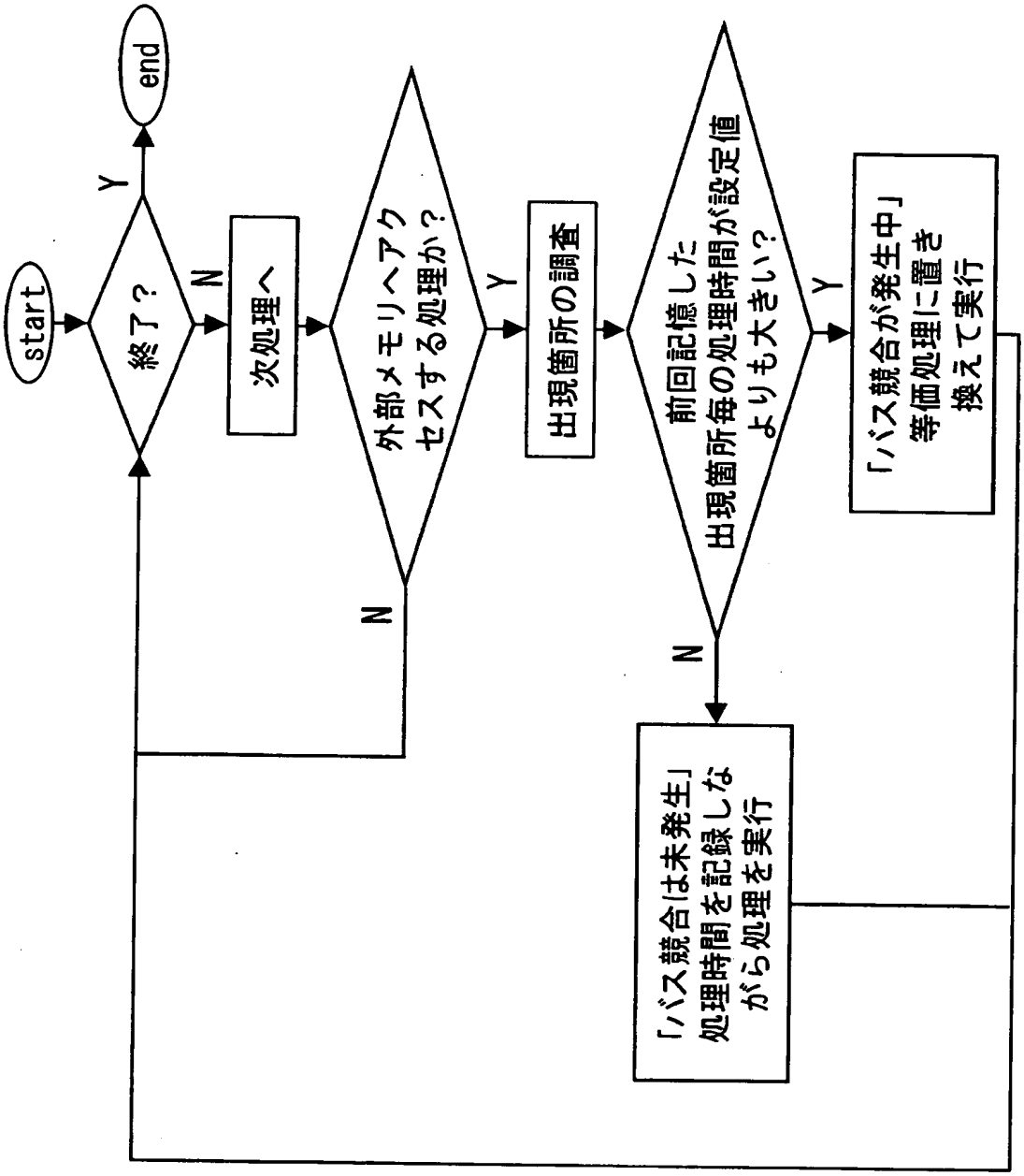
【図１２】


start
終了？
Y
end
N
次処理へ
外部メモリへアクセスする処理か？
Y
N
前回記憶した処理時間が設定値よりも大きい？
Y
N
「バス競合が発生中」等価処理に置き換えて実行
「バス競合は未発生」処理時間を記録しながら処理を実行

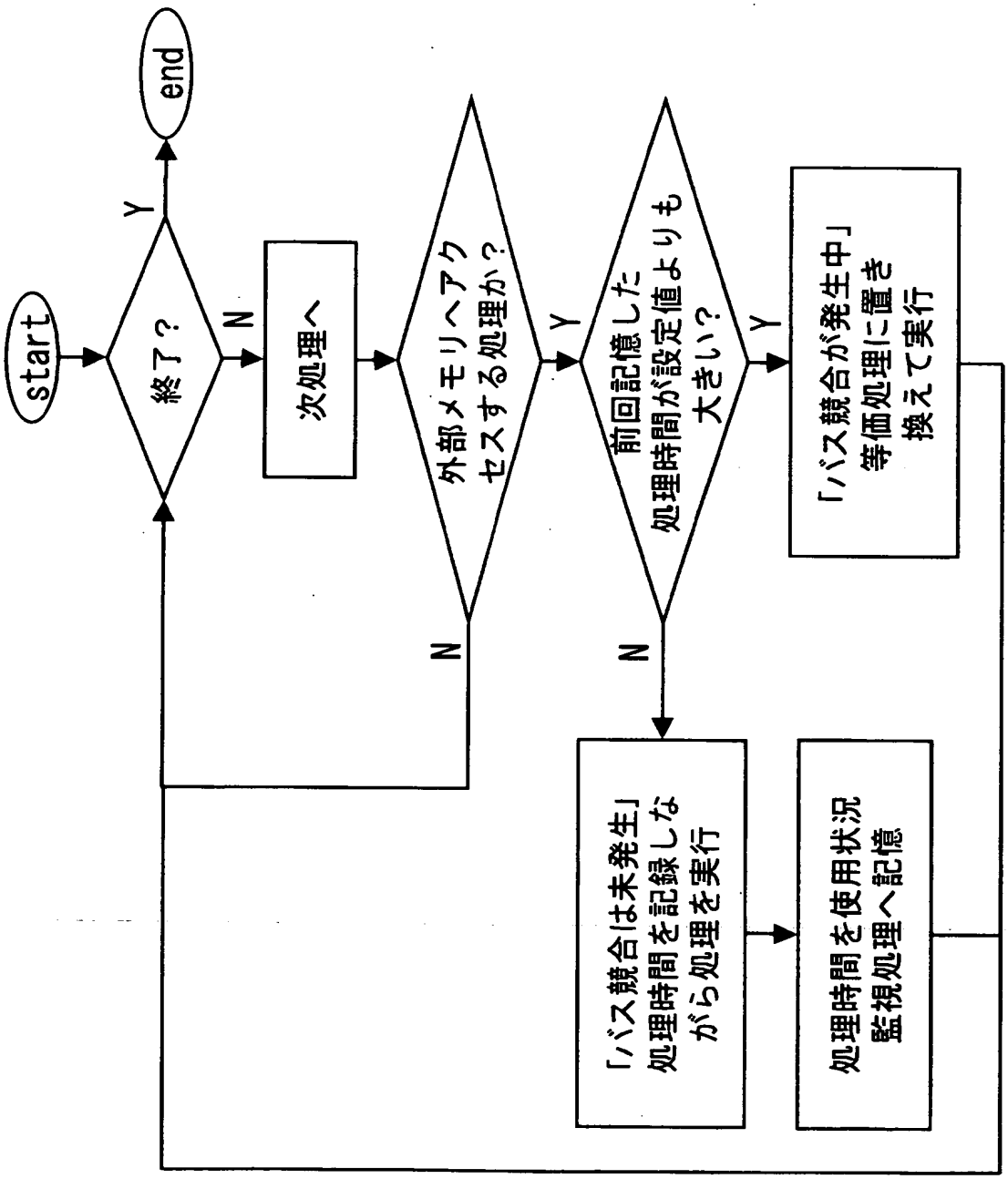
【図１３】

処理時間 設定値	等価処理へ置き換える確率[%]	等価処理へ置き換えない確率[%]
～2．0	60	40
2．0～1．8	70	30
1．8～1．6	80	20
1．6～1．4	90	10
1．4～1．2	95	5
1以下	0	100

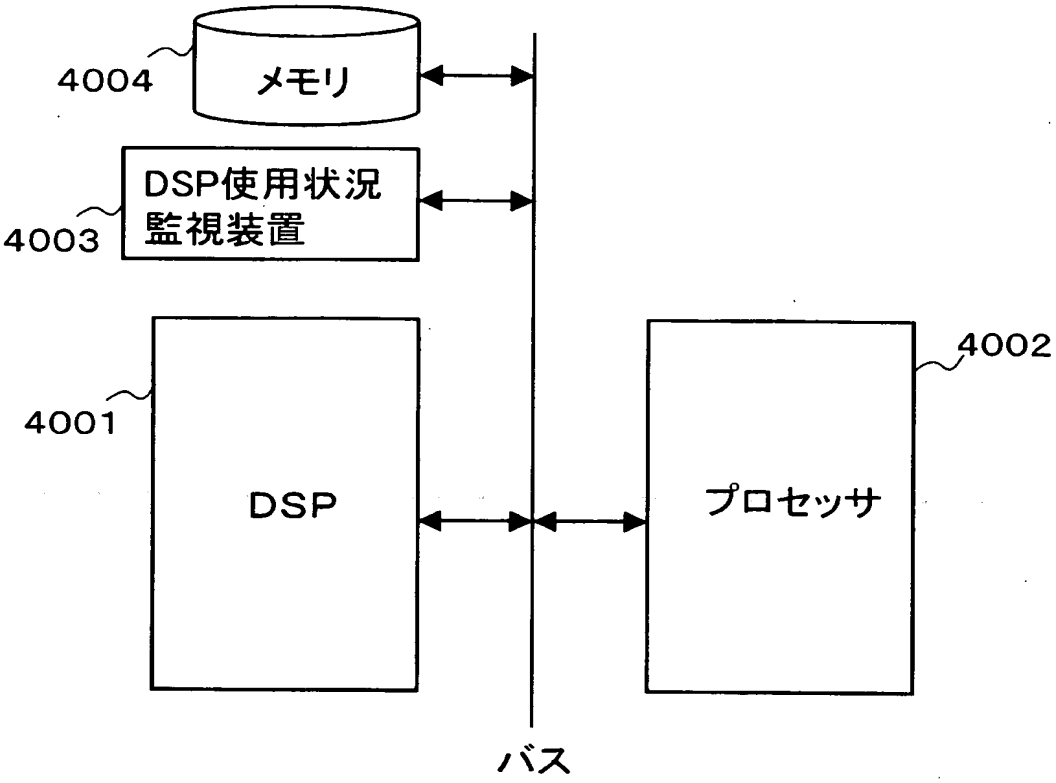
【図１４】


start
終了？
Y
end
N
次処理へ
外部メモリへアクセスする処理か？
Y
N
出現箇所の調査
前回記憶した出現箇所毎の処理時間が設定値よりも大きい？
Y
N
「バス競合が発生中」等価処理に置き換えて実行
「バス競合は未発生」処理時間を記録しながら処理を実行

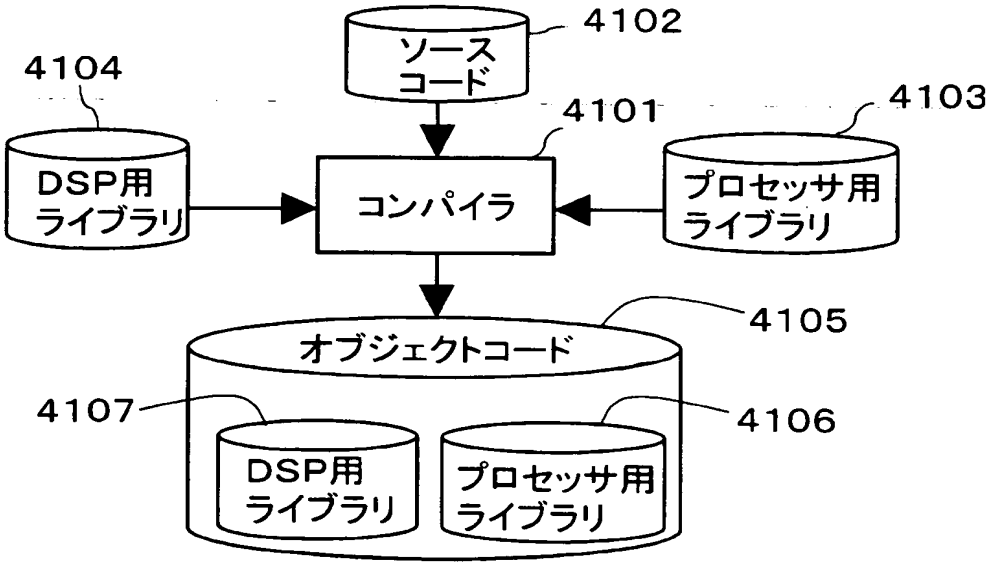
【図１５】


start
終了？
Y
end
N
次処理へ
外部メモリへアクセスする処理か？
Y
N
前回記憶した処理時間が設定値よりも大きい？
Y
N
「バス競合が発生中」等価処理に置き換えて実行
「バス競合は未発生」処理時間を記録しながら処理を実行
処理時間を使用状況監視処理へ記憶

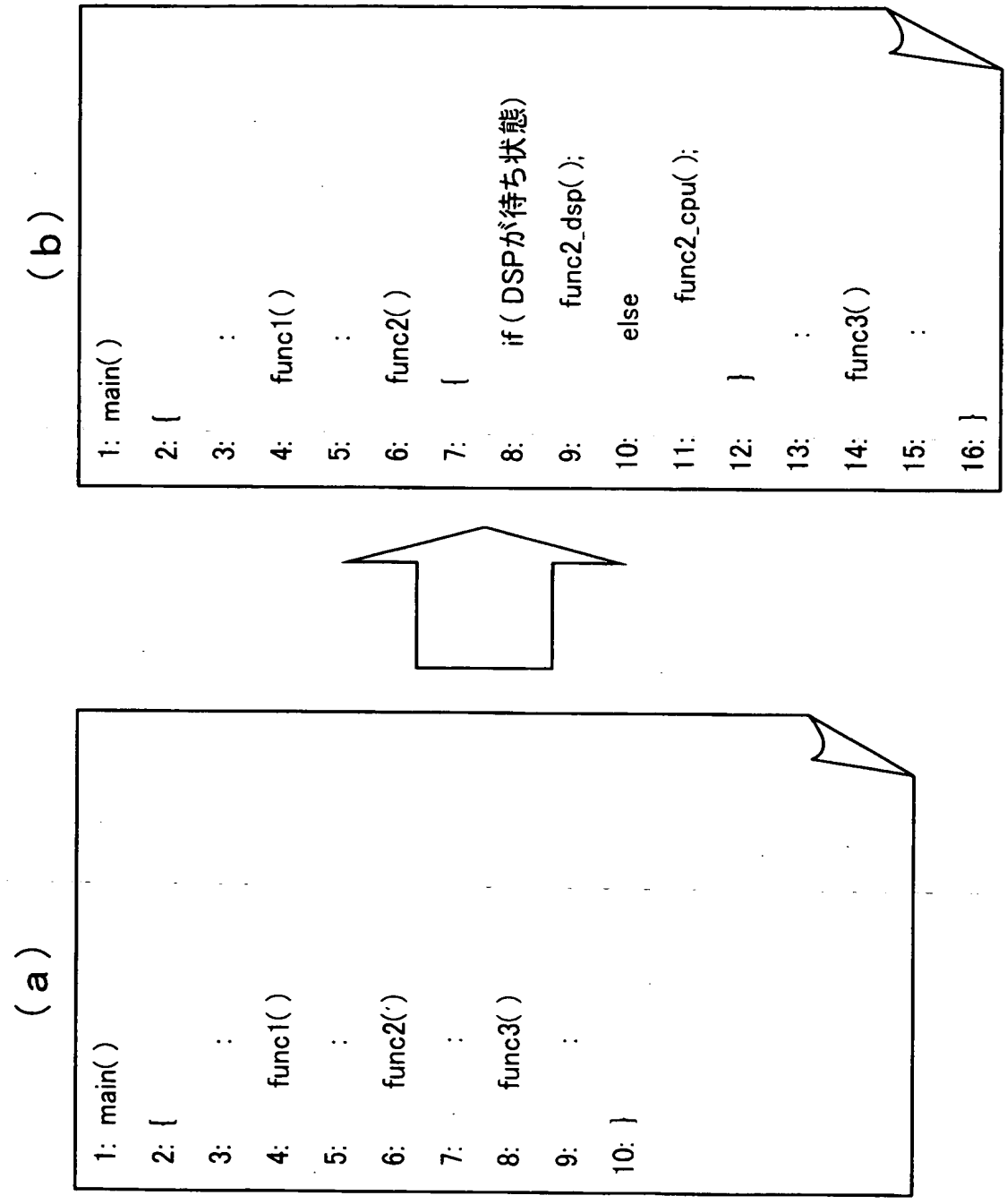
【図１６】


4004
メモリ
DSP使用状況
監視装置
4003
4001
DSP
4002
プロセッサ
バス

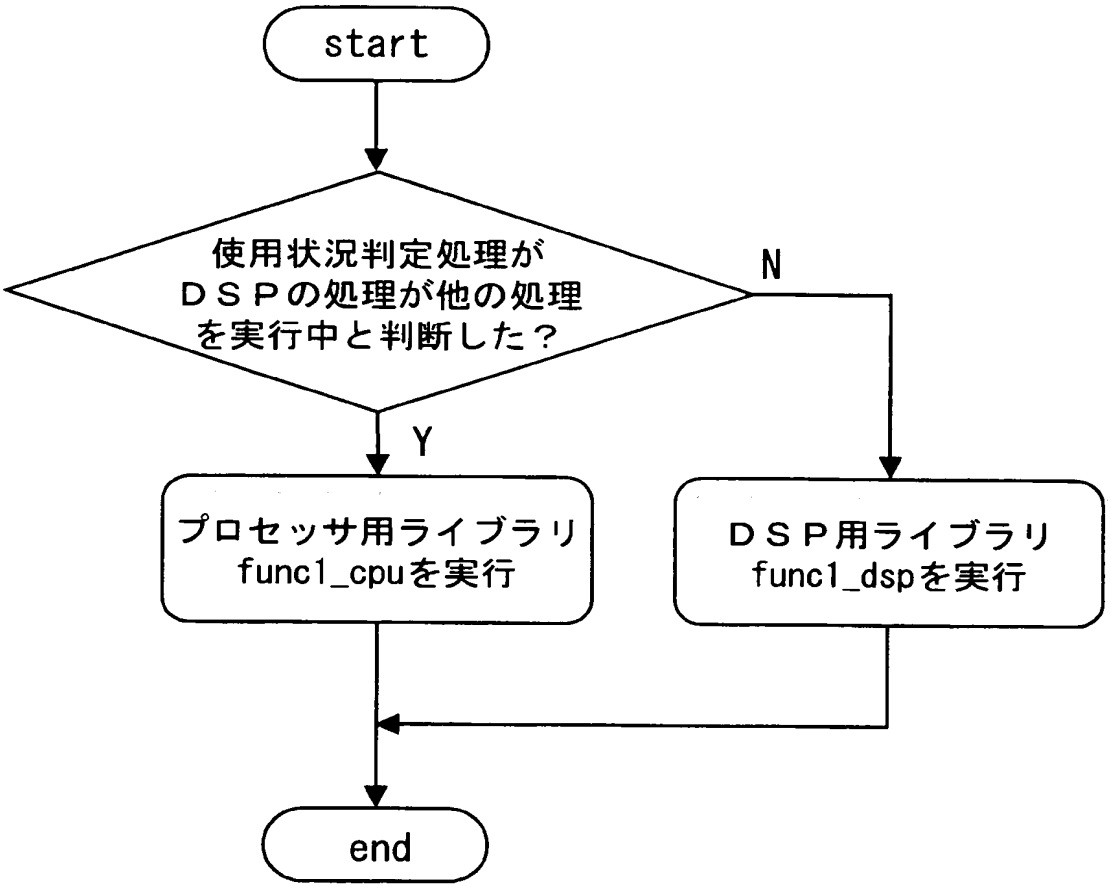
【図１７】


4102
ソース
コード
4104
4101
4103
DSP用
ライブラリ
コンパイラ
プロセッサ用
ライブラリ
4105
オブジェクトコード
4107
4106
DSP用
ライブラリ
プロセッサ用
ライブラリ

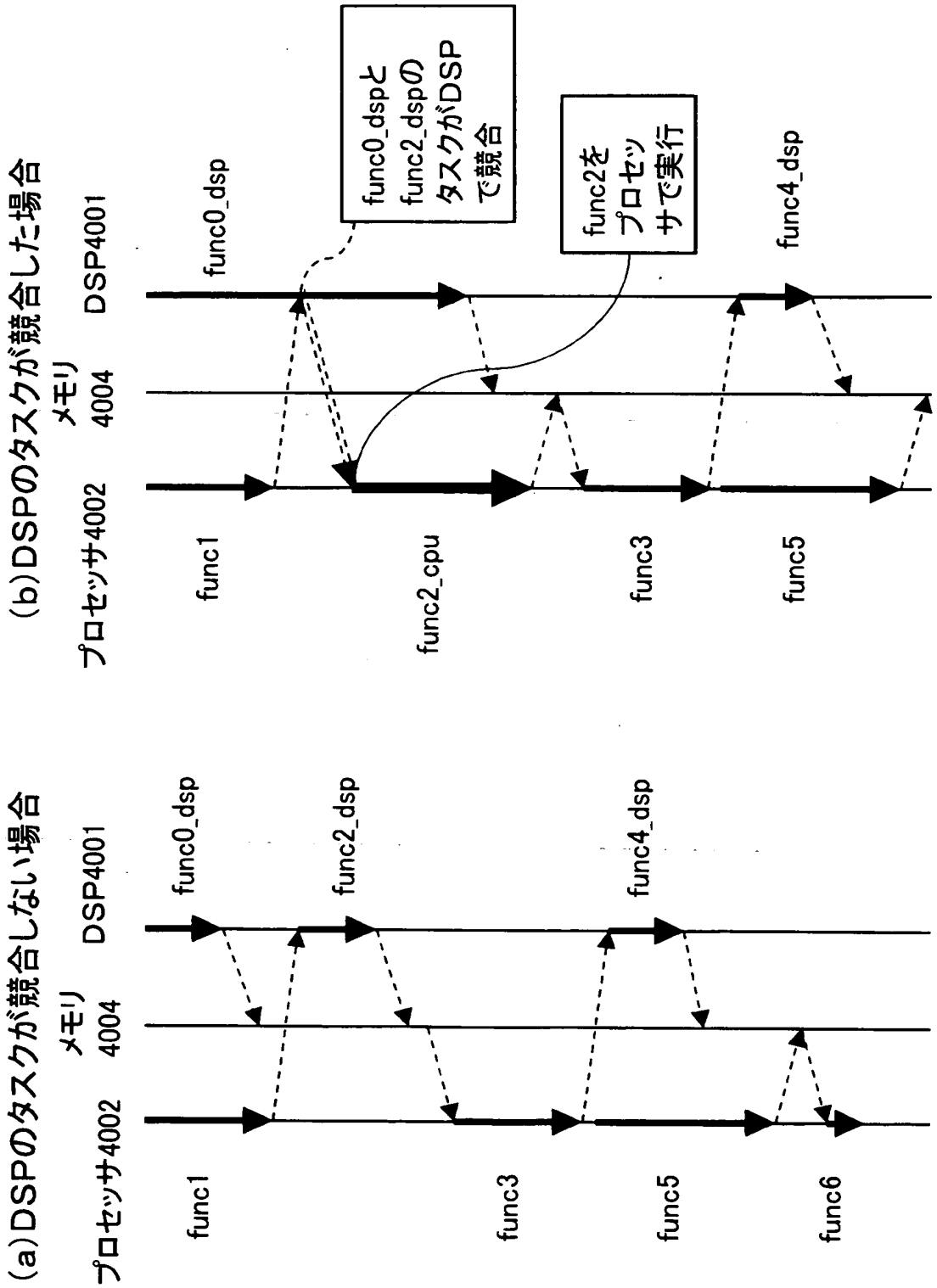
【図１８】


(a)
1: main( )
2: {
3: :
4: func1( )
5: :
6: func2( )
7: :
8: func3( )
9: :
10: }
(b)
1: main( )
2: {
3: :
4: func1( )
5: :
6: func2( )
7: {
8: if (DSPが待ち状態)
9: func2_dsp( );
10: else
11: func2_cpu( );
12: }
13: :
14: func3( )
15: :
16: }

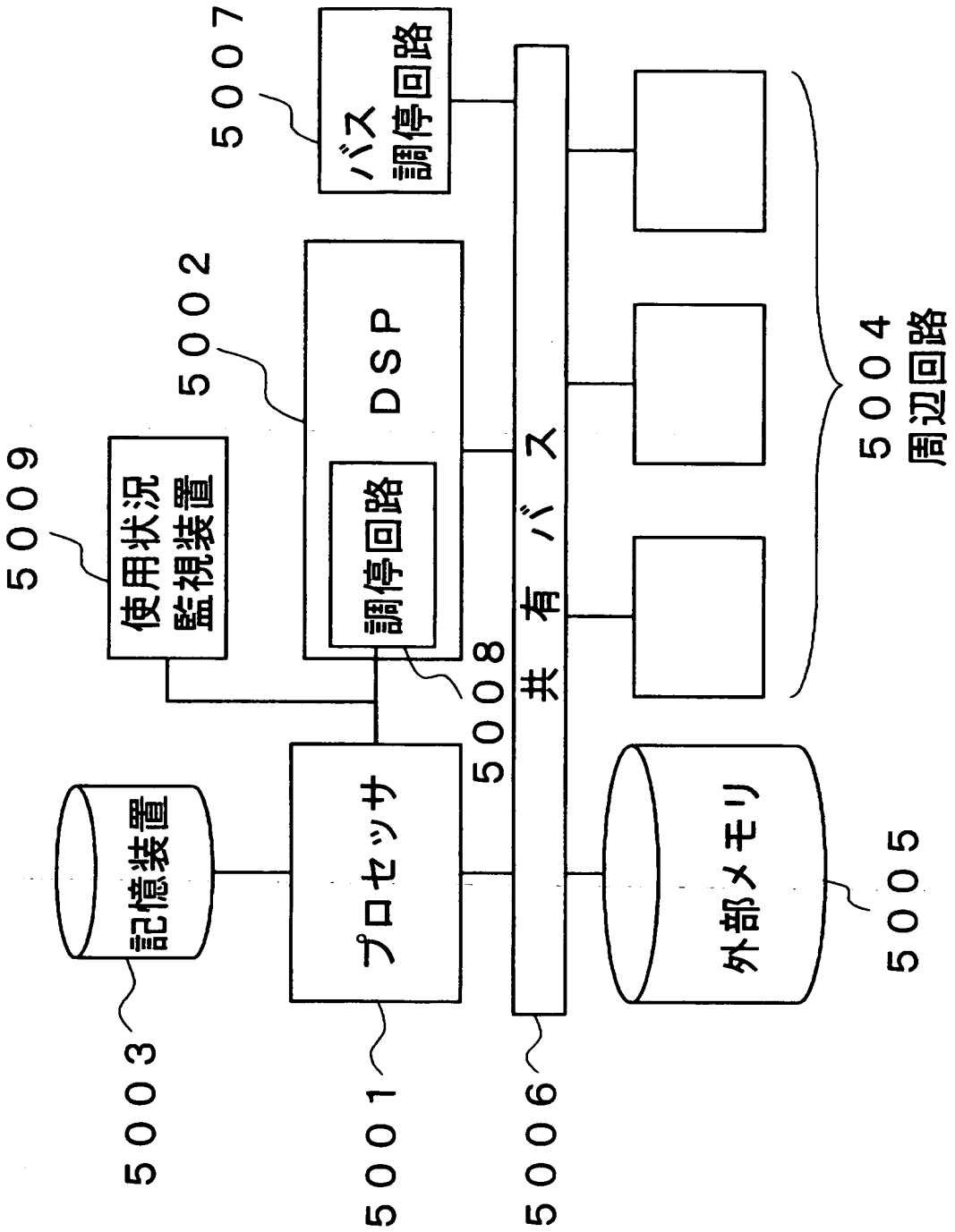
【図１９】


start
使用状況判定処理が
ＤＳＰの処理が他の処理
を実行中と判断した？
N
Y
プロセッサ用ライブラリ
func1_cpuを実行
ＤＳＰ用ライブラリ
func1_dspを実行
end

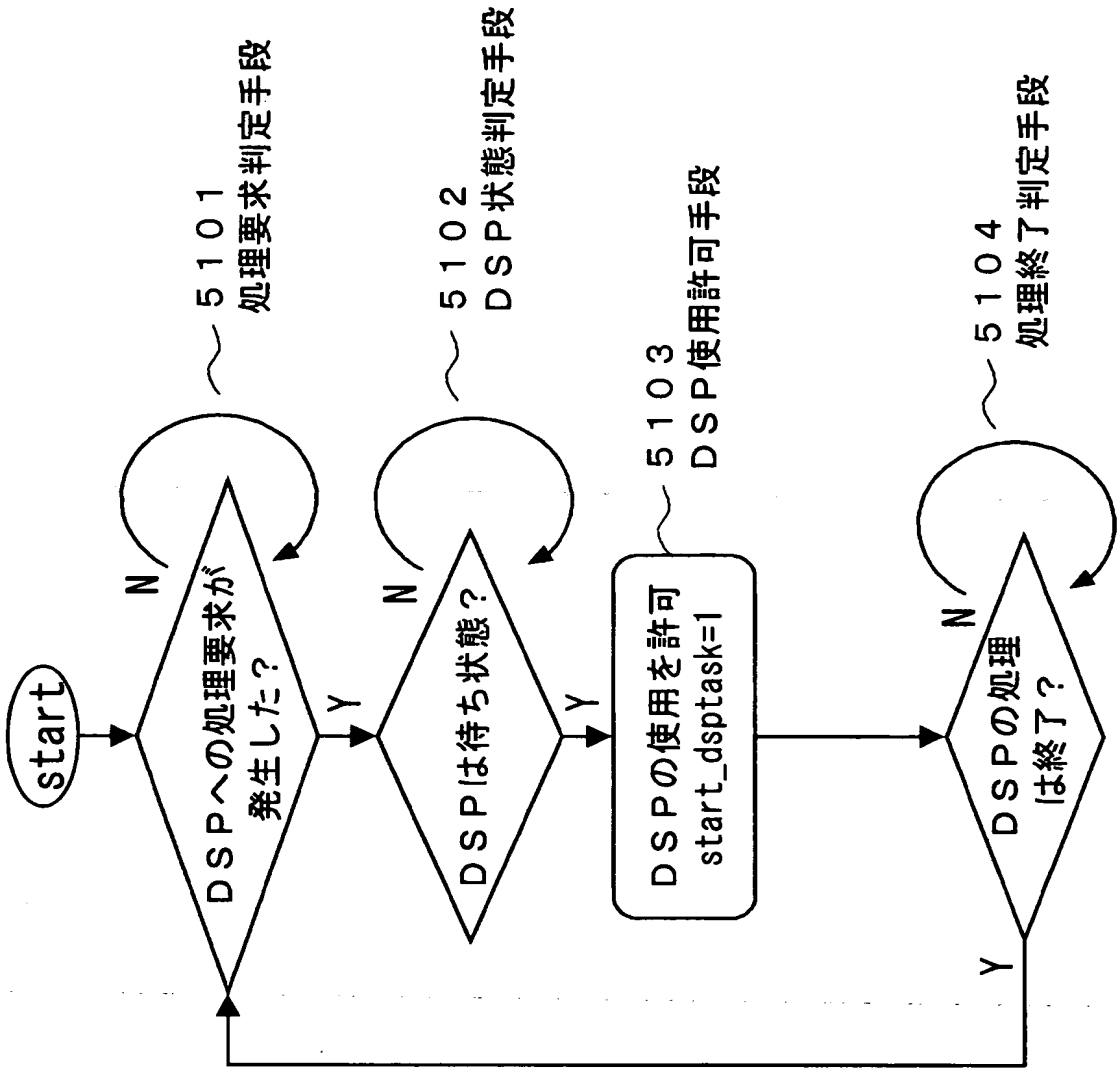
【図２０】


(a)DSPのタスクが競合しない場合
プロセッサ4002
メモリ
4004
DSP4001
func0_dsp
func2_dsp
func4_dsp
func1
func3
func5
func6
(b)DSPのタスクが競合した場合
プロセッサ4002
メモリ
4004
DSP4001
func0_dsp
func0_dspと
func2_dspの
タスクがDSP
で競合
func2を
プロセッ
サで実行
func4_dsp
func1
func2_cpu
func3
func5

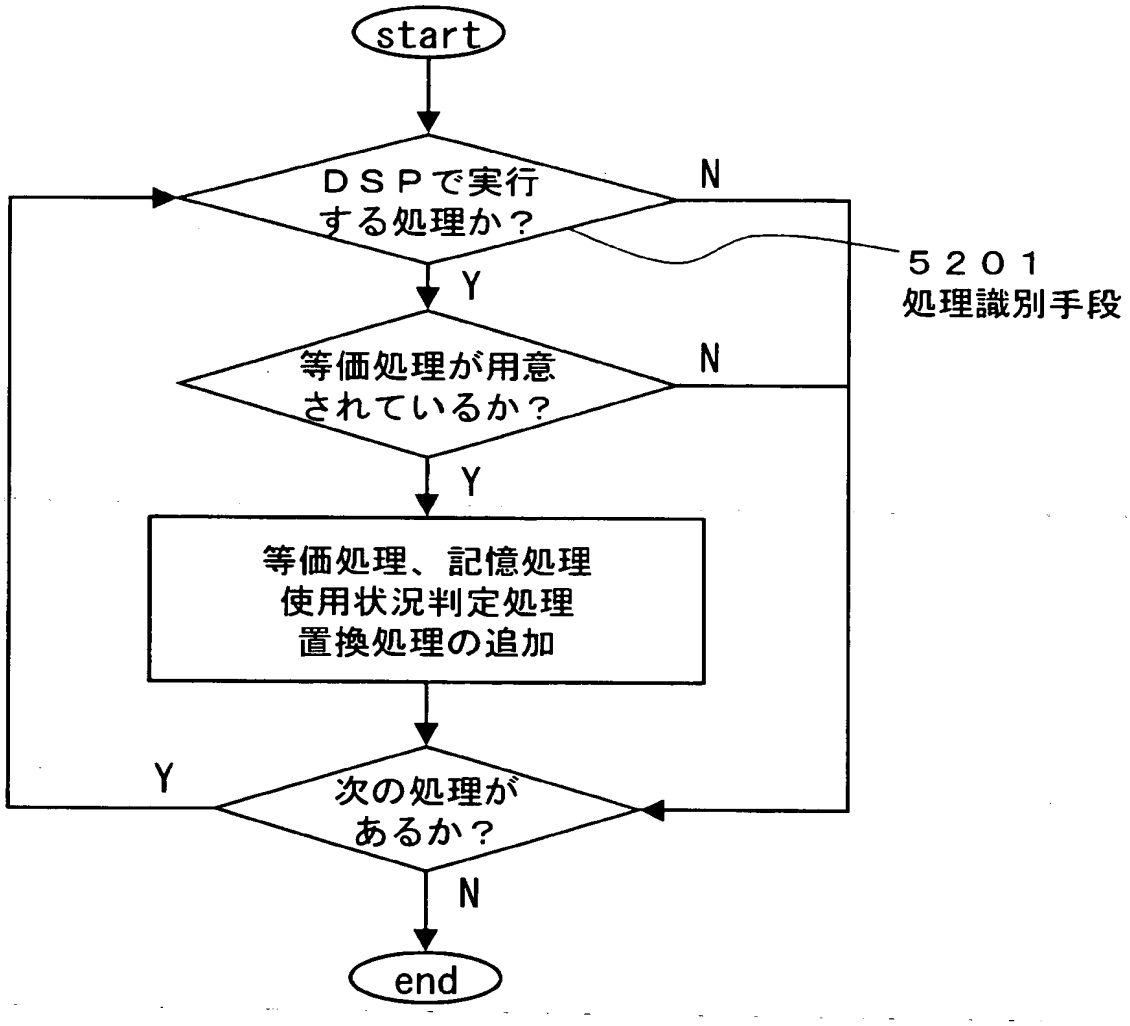
【図２１】


5007
バス調停回路
5002
DSP
調停回路
5009
使用状況監視装置
5008
共有バス
5004
周辺回路
記憶装置
プロセッサ
外部メモリ
5005
5003
5001
5006

【図２２】


start
DSPへの処理要求が発生した？
N
Y
5101
処理要求判定手段
DSPは待ち状態？
N
Y
5102
DSP状態判定手段
DSPの使用を許可
start_dsptask=1
5103
DSP使用許可手段
DSPの処理は終了？
N
Y
5104
処理終了判定手段

【図２３】


start
ＤＳＰで実行する処理か？
N
Y
５２０１
処理識別手段
等価処理が用意されているか？
N
Y
等価処理、記憶処理
使用状況判定処理
置換処理の追加
Y
次の処理があるか？
N
end

【図２４】

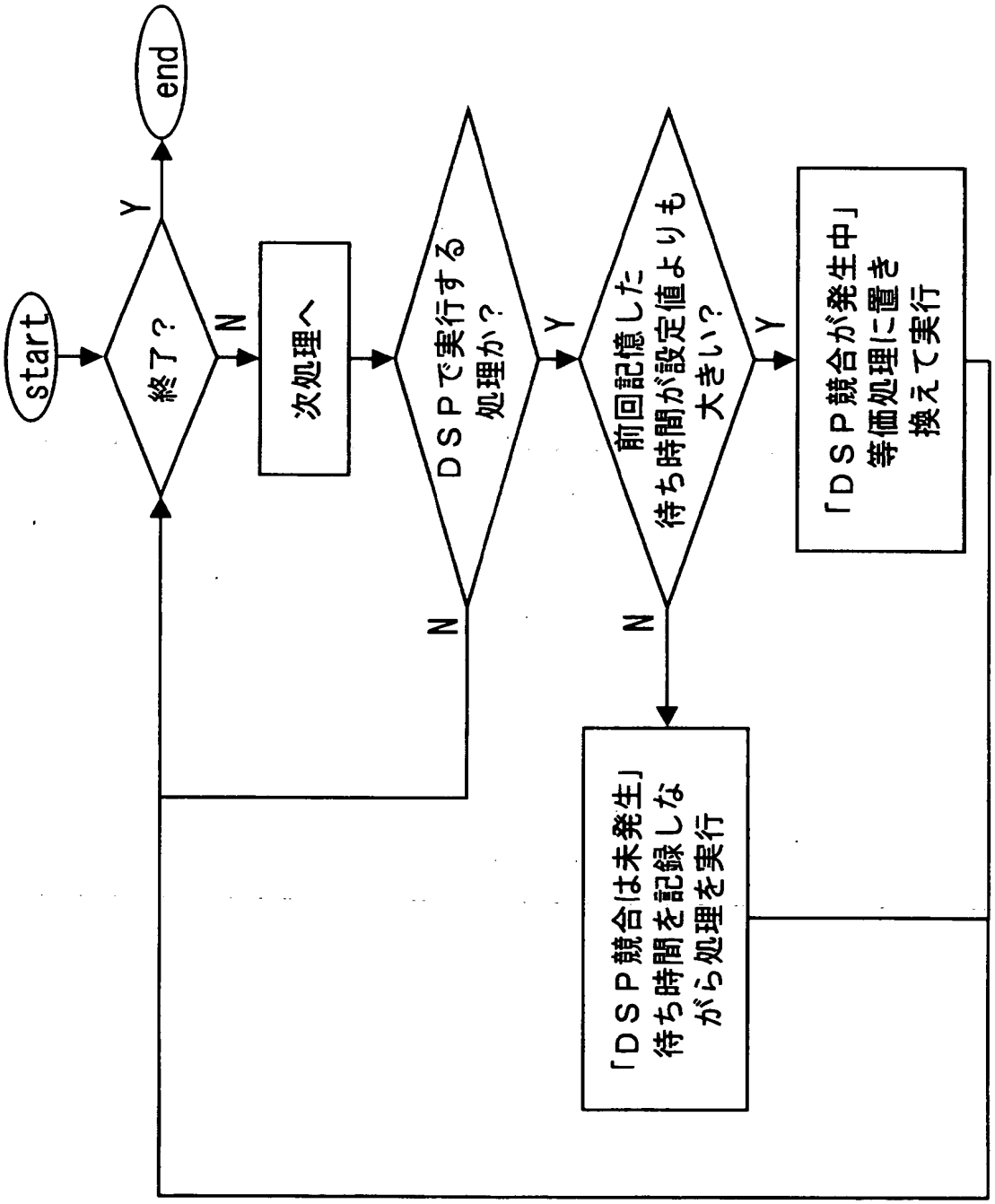
```
1: proc_task1() {
2:       dsp_task();
3: };
```

(a)コンパイラ実行前

```
1: proc_task1() {
2:     if( wait_time < DEFINED_TIME ) {
3:         start_time = timer_count;
4:         (DSPが処理を開始できるまで待つ)
5:         end_time   = timer_count;
6:         wait_time = end_time - start_time;
7:         dsp_task();
8:     }
9:     else{
10:        proc_dsptask();
11:    };
12: };
```

(b)コンパイラ実行後

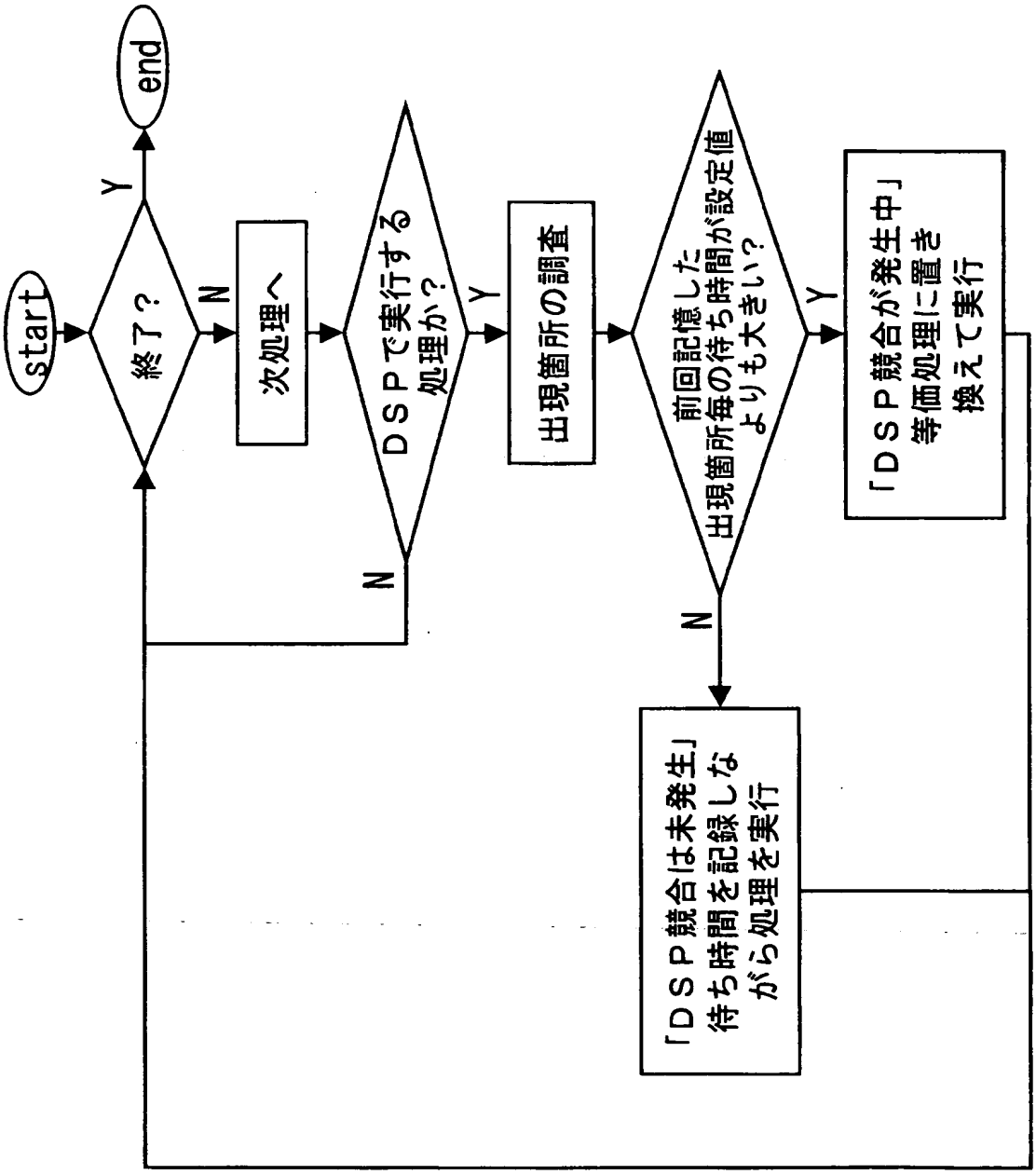
【図２５】


start
終了？
Y
end
N
次処理へ
DSPで実行する処理か？
Y
N
前回記憶した待ち時間が設定値よりも大きい？
Y
N
「DSP競合が発生中」等価処理に置き換えて実行
「DSP競合は未発生」待ち時間を記録しながら処理を実行

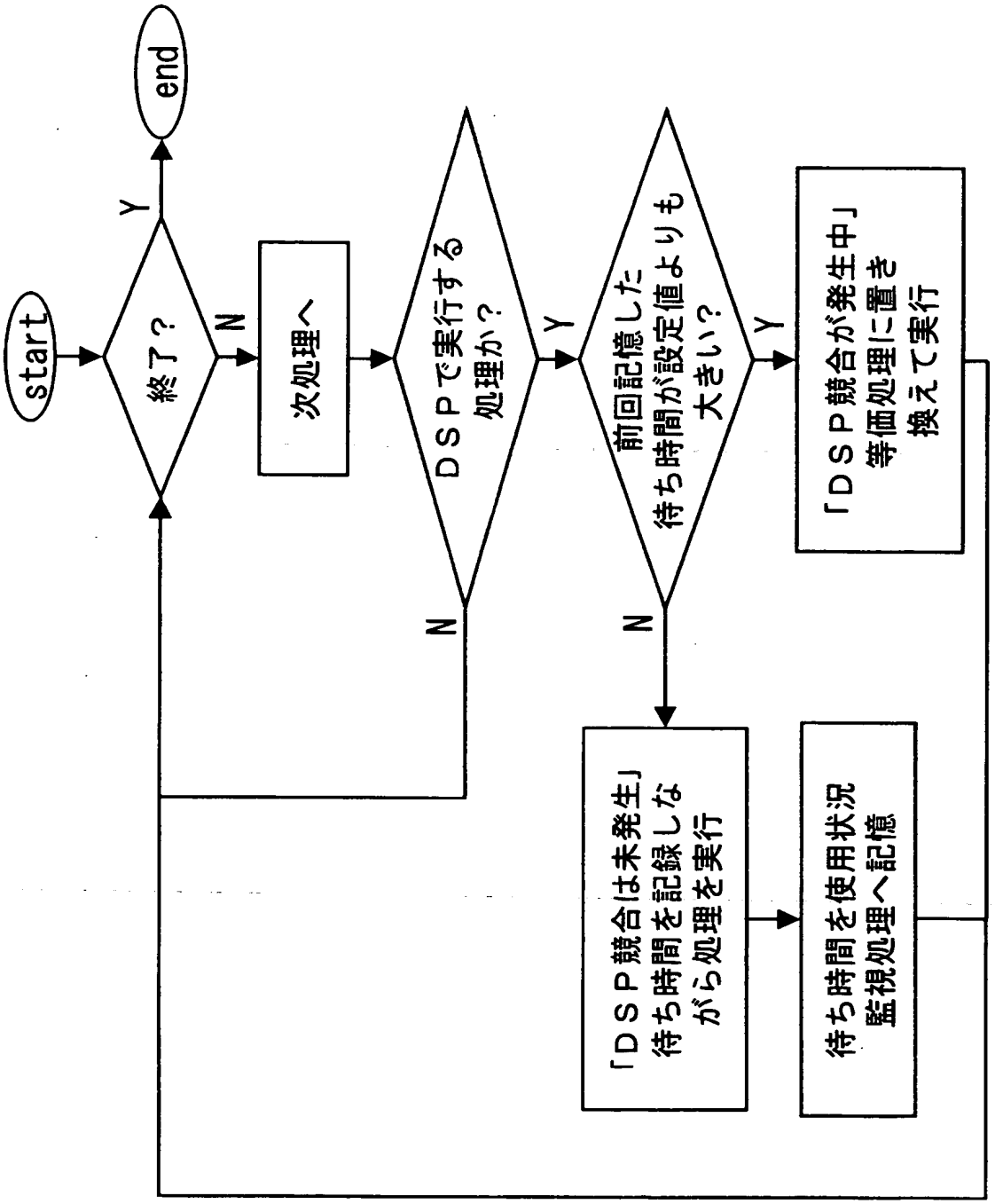
【図２６】

待ち時間 設定値	等価処理へ置き 換える確率[%]	等価処理へ置き 換えない確率[%]
～2．0	60	40
2．0～1．8	70	30
1．8～1．6	80	20
1．6～1．4	90	10
1．4～1．2	95	5
1以下	0	100

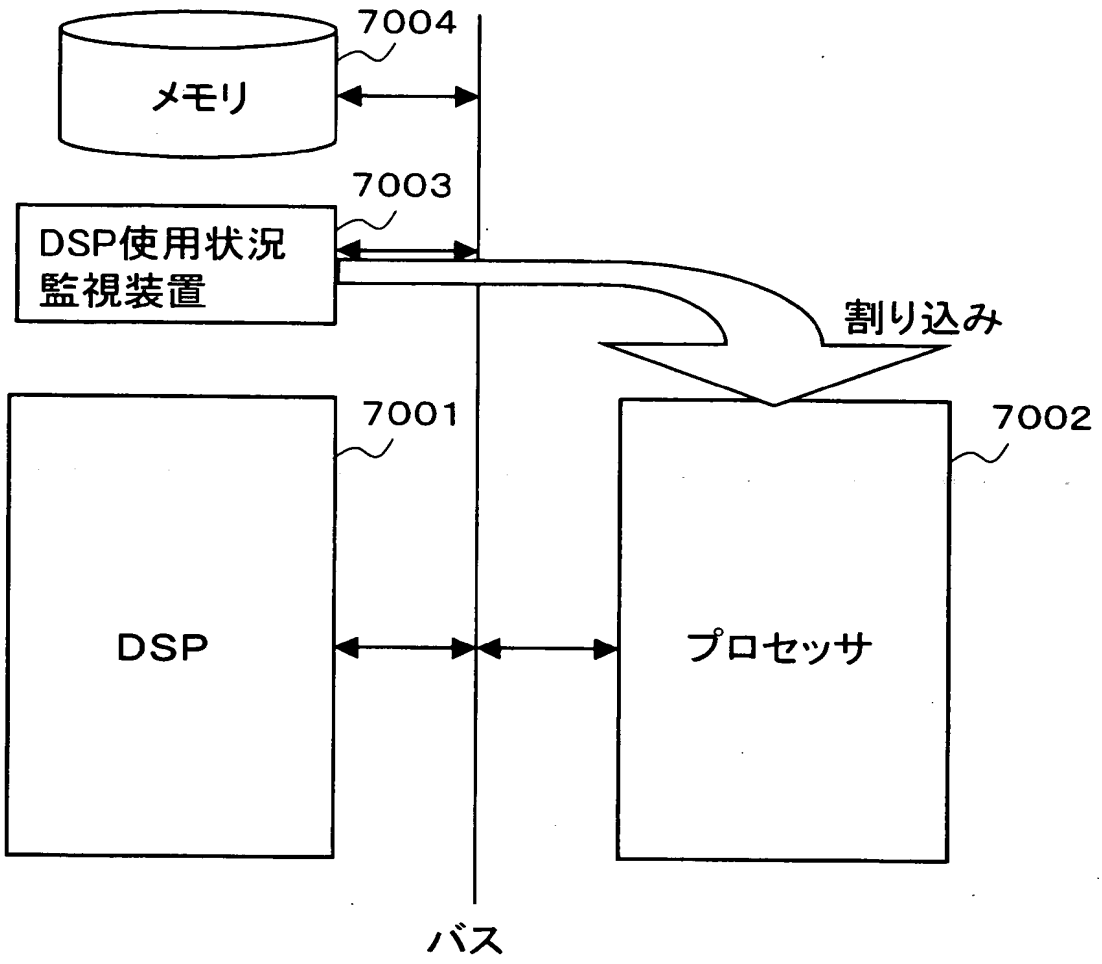
【図２７】


start
終了？
Y
end
N
次処理へ
DSPで実行する処理か？
N
Y
出現箇所の調査
前回記憶した出現箇所毎の待ち時間が設定値よりも大きい？
N
Y
「DSP競合は未発生」待ち時間を記録しながら処理を実行
「DSP競合が発生中」等価処理に置き換えて実行

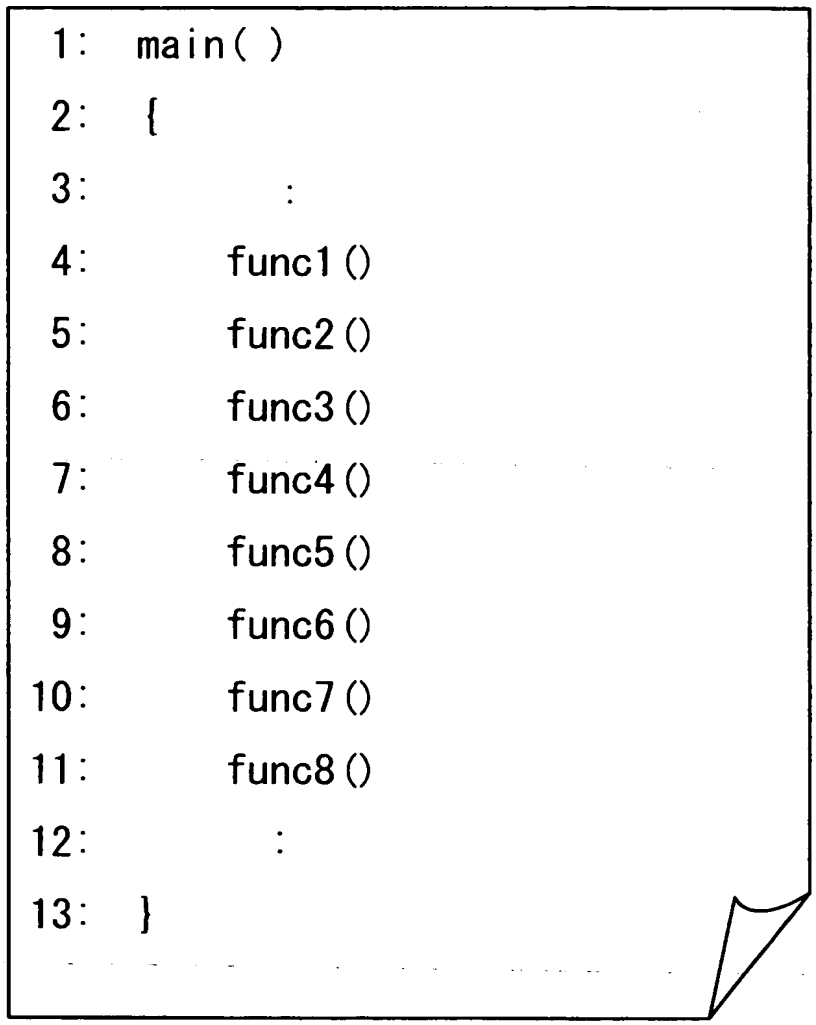
【図２８】


start
終了？
Y
end
N
次処理へ
DSPで実行する処理か？
N
Y
前回記憶した待ち時間が設定値よりも大きい？
N
Y
「DSP競合が発生中」等価処理に置き換えて実行
「DSP競合は未発生」待ち時間を記録しながら処理を実行
待ち時間を使用状況監視処理へ記憶

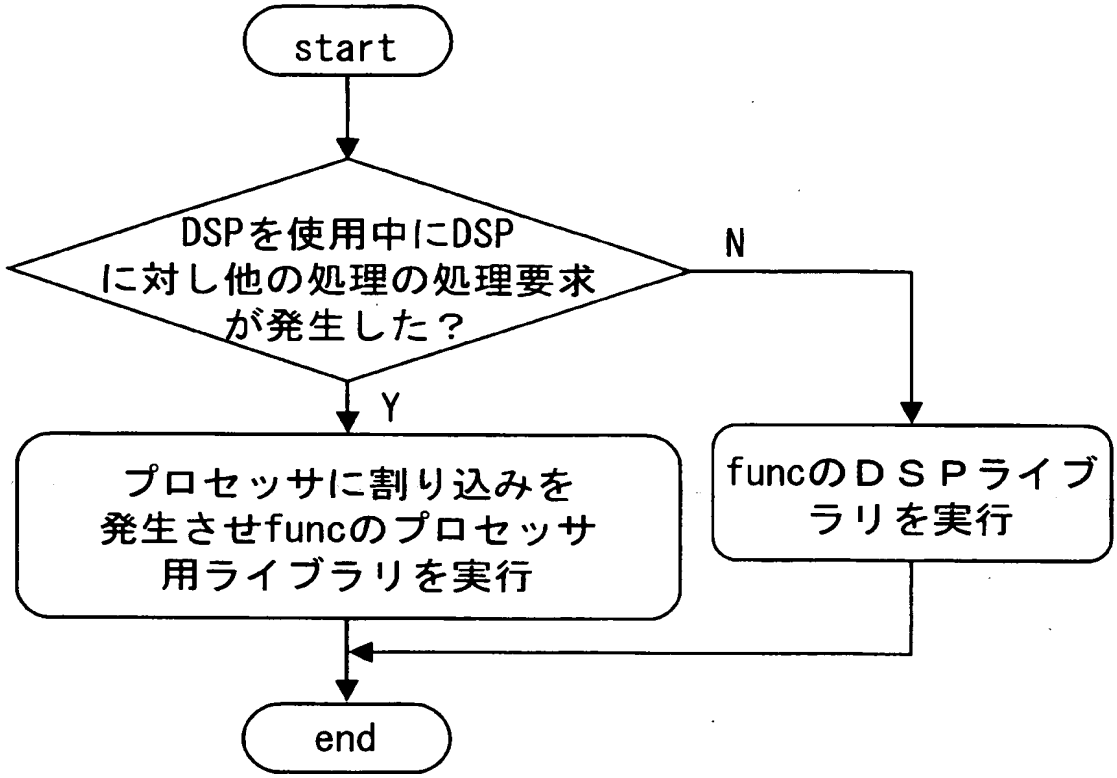
【図２９】



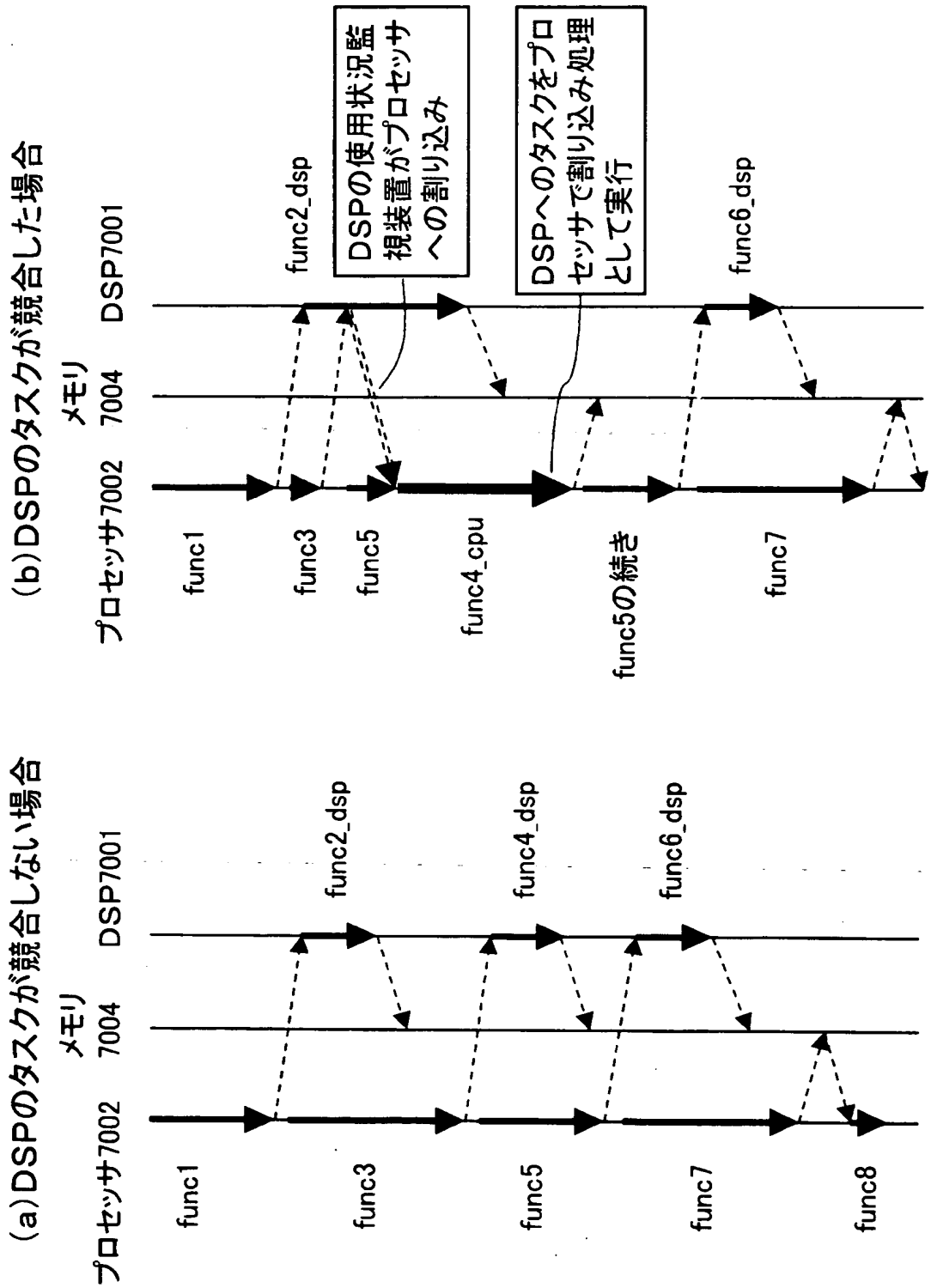
【図３０】


1: main( )
2: {
3: :
4: func1 ()
5: func2 ()
6: func3 ()
7: func4 ()
8: func5 ()
9: func6 ()
10: func7 ()
11: func8 ()
12: :
13: }

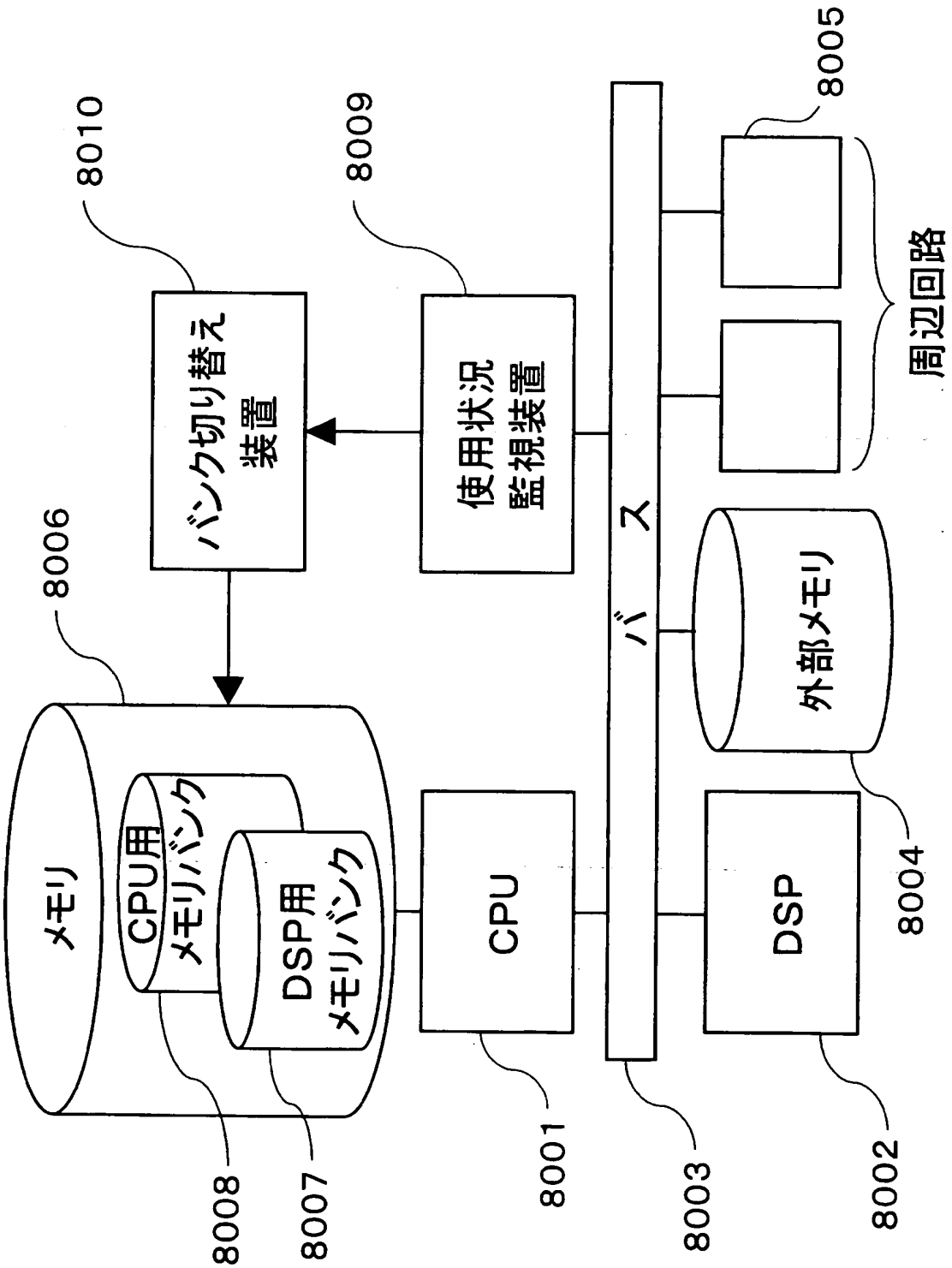
【図３１】


start
DSPを使用中にDSP
に対し他の処理の処理要求
が発生した？
N
Y
プロセッサに割り込みを
発生させfuncのプロセッサ
用ライブラリを実行
funcのＤＳＰライブ
ラリを実行
end

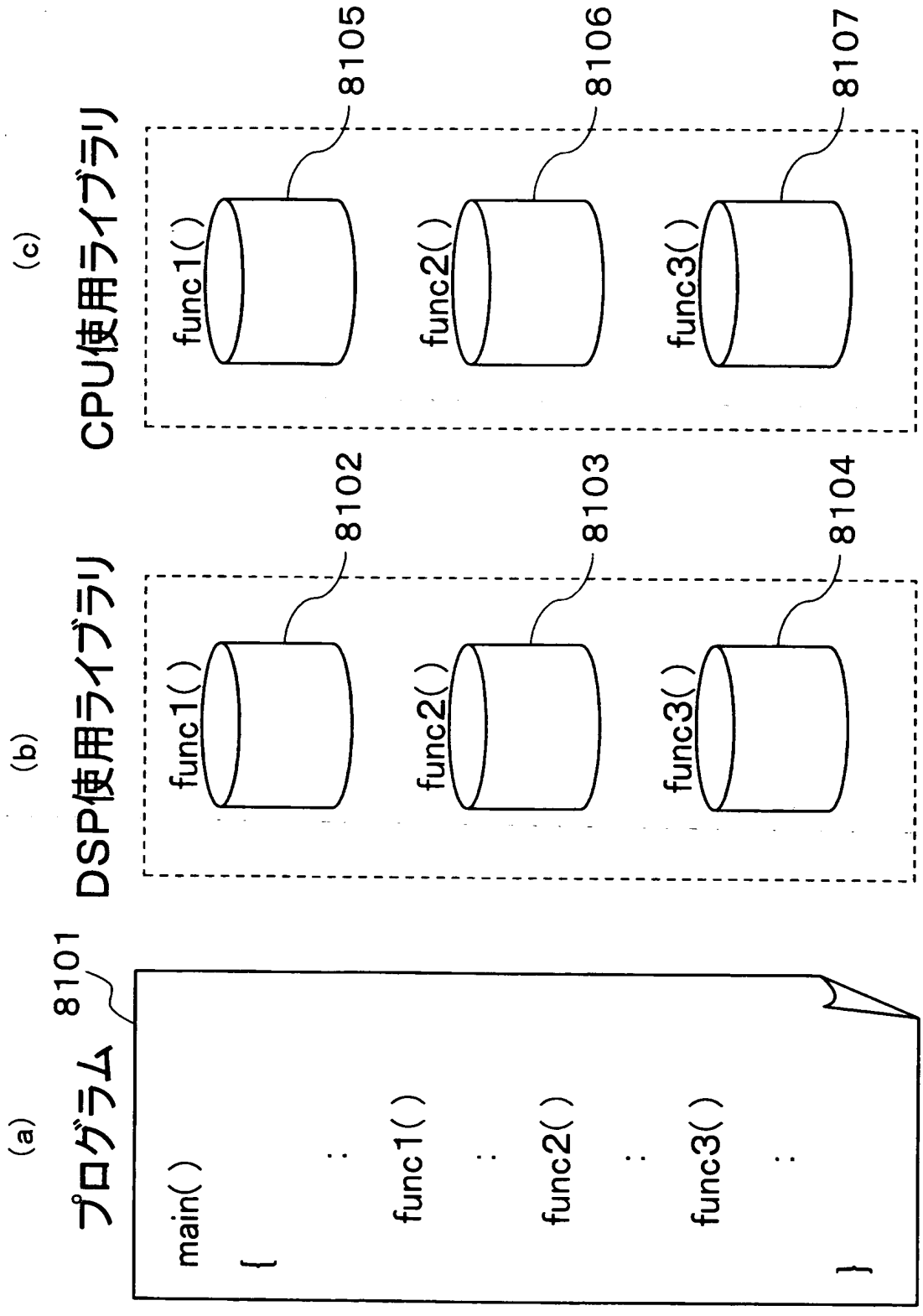
【図３２】


(a)DSPのタスクが競合しない場合
プロセッサ7002
メモリ 7004
DSP7001
func1
func2_dsp
func3
func4_dsp
func5
func6_dsp
func7
func8
(b)DSPのタスクが競合した場合
プロセッサ7002
メモリ 7004
DSP7001
func1
func2_dsp
DSPの使用状況監視装置がプロセッサへの割り込み
func3
func5
func4_cpu
DSPへのタスクをプロセッサで割り込み処理として実行
func5の続き
func6_dsp
func7

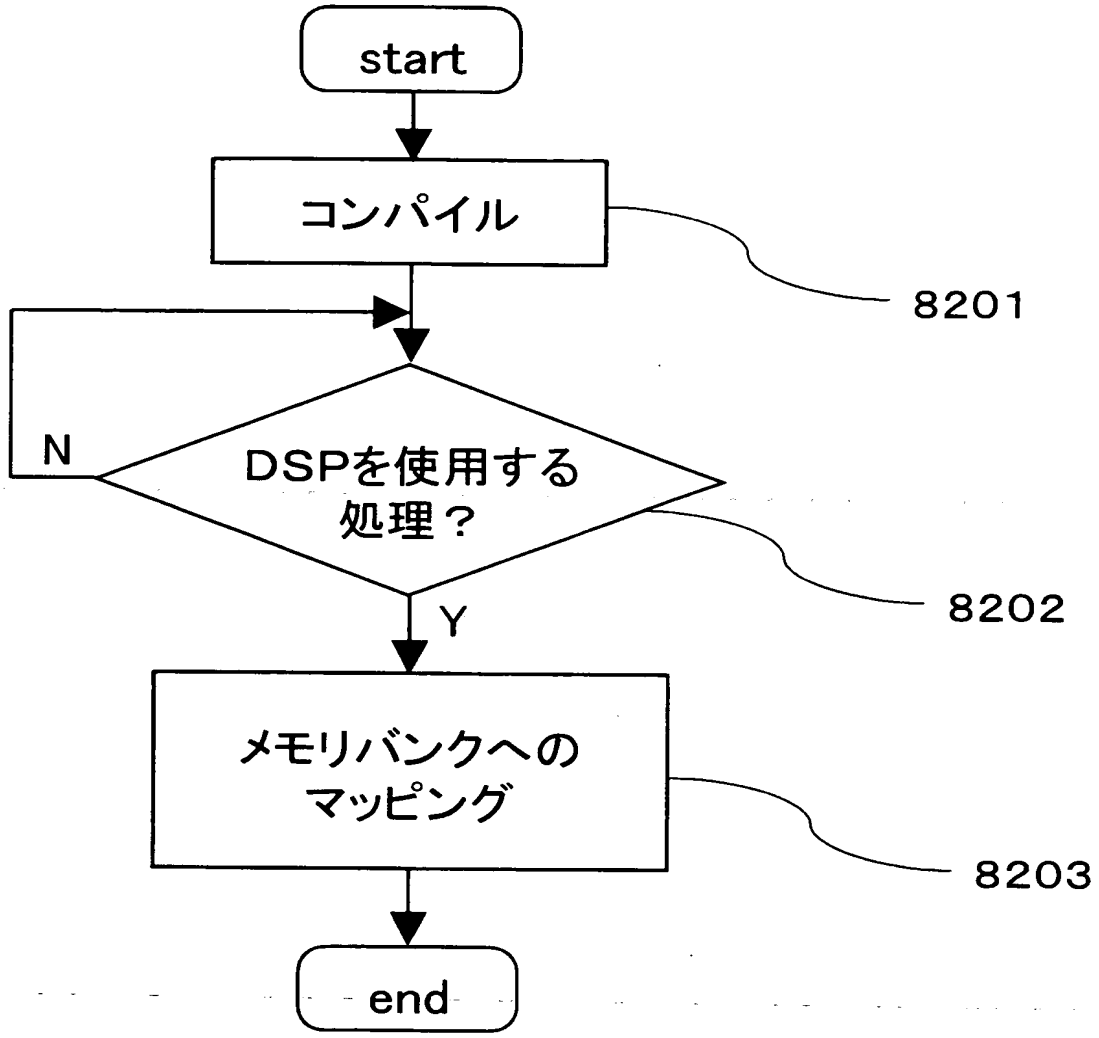
【図３３】


8010
8009
8005
バンク切り替え装置
使用状況監視装置
バス
周辺回路
8006
メモリ
CPU用メモリバンク
DSP用メモリバンク
CPU
外部メモリ
DSP
8004
8008
8007
8001
8003
8002

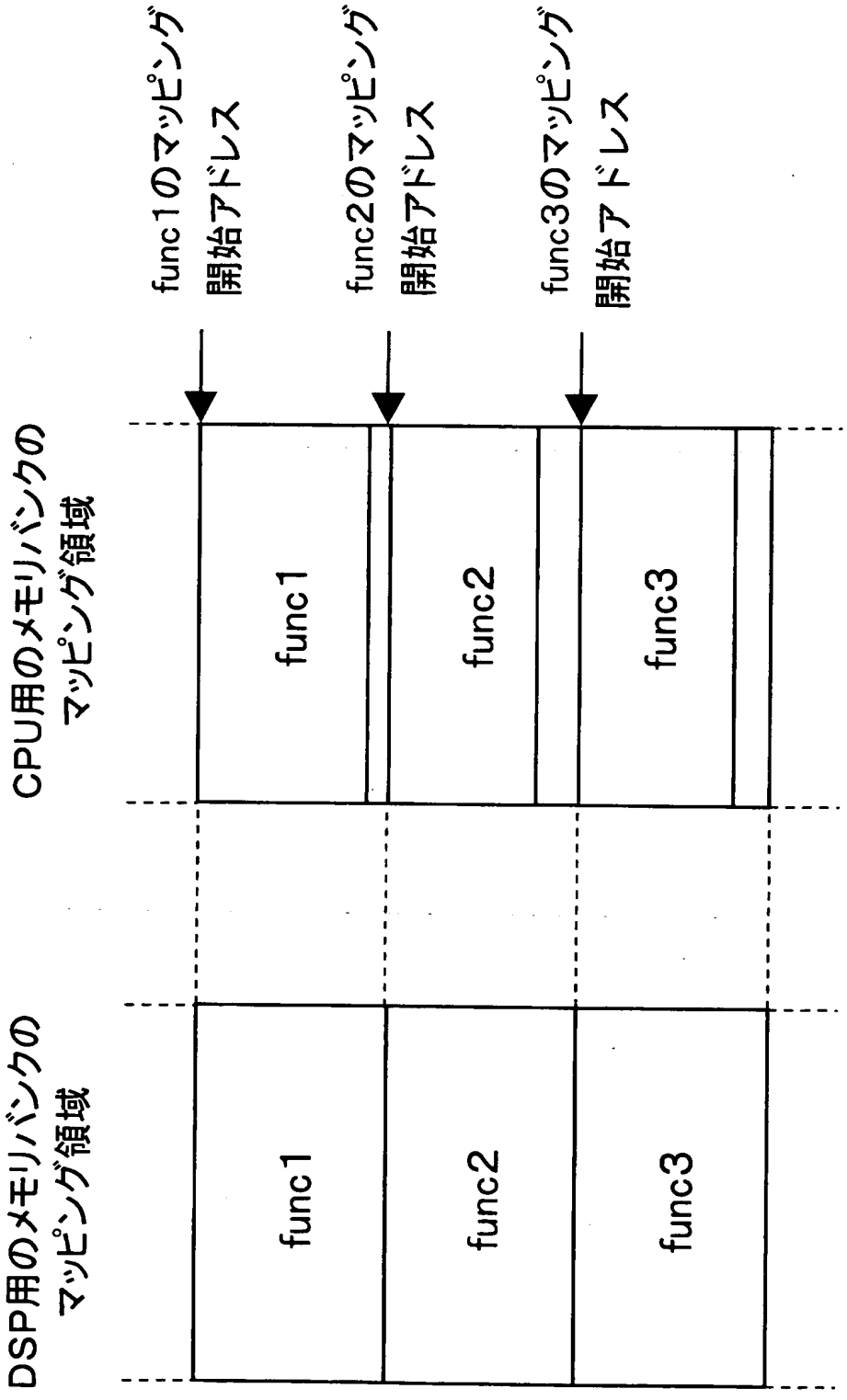
【図３４】


(a)
プログラム
8101
main( )
{
..
func1( )
..
func2( )
..
func3( )
..
}
(b)
DSP使用ライブラリ
8102
8103
8104
func1( )
func2( )
func3( )
(c)
CPU使用ライブラリ
8105
8106
8107
func1( )
func2( )
func3( )

【図３５】


start
コンパイル
8201
N
DSPを使用する
処理？
8202
Y
メモリバンクへの
マッピング
8203
end

【図３６】


func1のマッピング
開始アドレス
func2のマッピング
開始アドレス
func3のマッピング
開始アドレス
CPU用のメモリバンクの
マッピング領域
func1
func2
func3
DSP用のメモリバンクの
マッピング領域
func1
func2
func3

【書類名】　要約書

【要約】

【課題】　従来、バスやＤＳＰなどの共有リソースを有するシステムにおいて、リソースを使用するソフトウェア処理が行われる際に、バスアクセス競合やＤＳＰ競合などのリソースの競合が頻繁に発生すると、結果としてシステム全体の性能が低下するという影響があった。本発明はかかる点に鑑み、リソースの競合の情報を考慮したソフトウェア処理システムを提供することを目的とする。

【解決手段】　リソースの使用状況を監視する使用状況監視処理の過程と、使用状況監視処理の過程の出力の情報に応じて実行されるソフトウェアの処理方法を変更するソフトウェア処理変更処理の過程を、ハードウェアまたはソフトウェアとして実現して、リソースの競合を発生しにくくしたものである。

【選択図】　図１

認定・付加情報

特許出願の番号	特願２００２－３５５６８３
受付番号	５０２０１８５３７０５
書類名	特許願
担当官	第七担当上席　　００９６
作成日	平成１４年１２月１９日

＜認定情報・付加情報＞

【提出日】　平成14年12月 6日

次頁無

特願２００２－３５５６８３

出　願　人　履　歴　情　報

識別番号　　[０００００５８２１]

１．変更年月日　　１９９０年　８月２８日
[変更理由]　　新規登録
住　所　　大阪府門真市大字門真１００６番地
氏　名　　松下電器産業株式会社